



---

# Writing Device Drivers





---

## **PART ONE: Regular Device Drivers**





---

## Hardware Context

Computer I/O architectures are far more dependent upon bus structure than they are upon CPU type, and device drivers, oriented as they are towards I/O, must have intimate knowledge of the bus characteristics of the machines on which they are running. For example, many Multibus machines do not support vectored interrupts<sup>1</sup> and thus drivers for interrupt driven devices which are intended to run on Multibus machines must provide polling facilities. Fortunately, the Sun kernel provides facilities (described in the *Other Kernel/Driver Interfaces* section of the *Overall Kernel Context* chapter) by which a driver can determine the type of the machine upon which it's running.

### 2.1. Multibus Machines

#### Multibus Memory Address Space and I/O Address Space

The MC680X0 family of processors does all its I/O via a process known as "memory mapping." What this means is that the processor sees no difference between memory and peripheral devices — all input-output operations are performed by storing data and fetching data from the same memory space. The Multibus, on the other hand, was originally designed for processors, like those of the Intel 8080 family, which have two separate address spaces. Such processors have one kind of instruction for storing data in memory or fetching data from memory (instructions such as MOV), and another, different kind of instruction (such as IN and OUT) for transferring data to or from peripheral devices. Reflecting the architecture of such processors, the Multibus has two address spaces.

#### *Multibus memory space*

is used for memory or devices that look like memory. Many devices — commonly known as "memory mapped" devices — are designed to be accessed as memory, and drivers for such devices can "map" them into user virtual memory space and then perform device I/O by simply reading and writing the device's memory as part of normal address space. Such memory-mapped drivers tend to be quite simple, and so it's notable that devices not explicitly designed to be memory mapped can, under a restricted set of circumstances, be driven by memory mapping. The restrictions are,

---

<sup>1</sup> The Multibus itself, as it turns out, actually does support vectored interrupts, but not in a way that can reasonably be used with the MC680X0 family of processors.



however, fairly severe. Such drivers cannot, for example, have `xxioctl()` routines. See the *Mapping Devices Without Device Drivers* section of the *Driver Development Topics* manual for more details. The Sun-2 Color Board is a good example of a device that is designed to be memory mapped, and a listing of its driver can be found in the *Sample Driver Listings* appendix.

#### *Multibus I/O address space*

is another “space” entirely separate from normal memory. Typically used as an area to which device registers can be mapped, I/O space was originally introduced to keep such registers out of limited primary address space by providing a means of making peripherals, rather than system memory, respond to the bus whenever given I/O control lines were asserted by the CPU. (Such a setup also reduces hardware costs by keeping the number of address lines small.) Devices which have their control and status registers mapped to Multibus I/O address space are said to be “I/O mapped” devices.

The MC680X0 family, of course, no longer suffers the addressing limitations that made the dual-space architecture of the Multibus so attractive. The VMEbus, in similar regard, is no longer structured around separate “memory” and “I/O” spaces. (The term “I/O space” does continue to be used, from time to time, with reference to VMEbus-based systems and devices. Such use, however, is largely by way of analogy with Multibus systems, and it shouldn’t be taken too literally).

Be aware that generic Multibus memory space can be either 20-bit or a 24-bit. (Sun normally uses 20-bit Multibus memory addresses, though when a Multibus card is installed in a VMEbus system with a VMEbus/Multibus adapter, 24-bit addresses are used). In similar regard, a generic Multibus can provide either an 8-bit or 16-bit I/O space, and Sun uses only the 16-bit Multibus I/O space. Note, however, that some older Multibus boards accept only 8-bit Multibus I/O addresses.

Sun Multibus systems actually have four “address spaces,” corresponding to the four types of memory (each type has an identifying number associated with it, a number which is used by the MMU in computing PTE’s (Page Table Entries). See the *Sun-2 Address Mapping* section of the *Driver Development Topics* chapter for details. Though you will seldom deal with the on-board address spaces, you’re best off understanding what they are. The following table thus contains not only the two Multibus spaces, but the “on board” memory and I/O spaces as well. It’s within these spaces, resident on the CPU board itself, that SunOS is run.

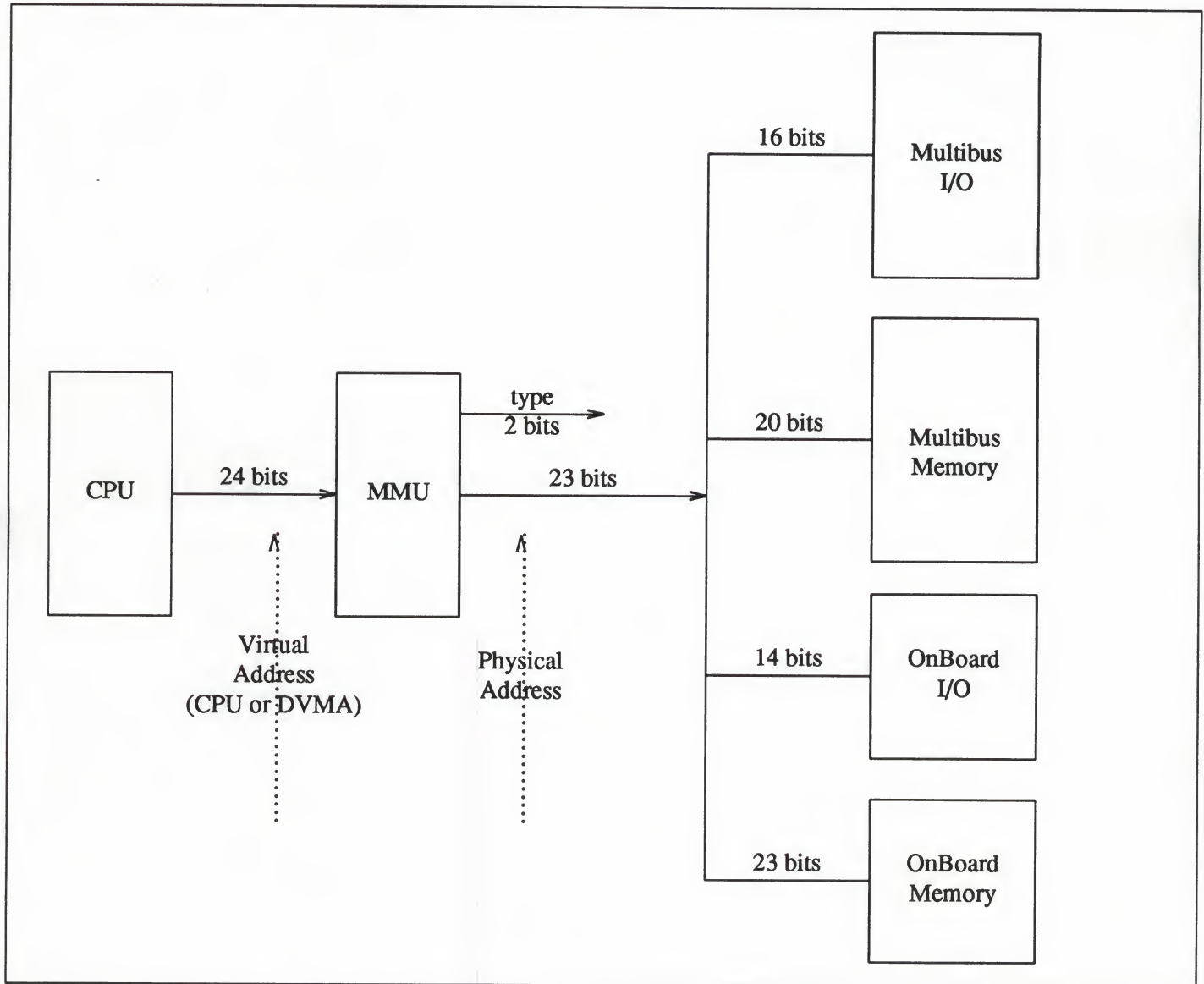
Table 2-1 *Sun-2 Multibus Memory Types*

| Type | Description        | Address Size | Address Range    |
|------|--------------------|--------------|------------------|
| 0    | On-Board Memory    | 23 bits      | 0x0 – 0x7FFFFFFF |
| 1    | On-Board I/O Space | 14 bits      | 0x0 – 0x3FFF     |
| 2    | Multibus Memory    | 20 bits      | 0x0 – 0xFFFFF    |
| 3    | Multibus I/O Space | 16 bits      | 0x0 – 0xFFFF     |



The following schematic view of the Sun-2 Multibus may help the driver developer to visualize the larger hardware context within which drivers operate (when running on a Sun-2 Multibus machine.)

Figure 2-1 *Sun-2 Multibus Address Spaces*



Note some significant aspects of addressing layout as indicated in this table.

- The Memory Management Unit is at the *center* of the picture, a position that reflects its importance in the addressing scheme of all Sun machines, VMEbus based as well as Multibus based. (The centrality of the MMU will become quite clear when you later set out to allocate a physical address to your device, and then examine/set it with the PROM monitor.)

- Secondly, the input address of the MMU is a 24-bit *virtual address*. It may originate with the CPU, or come from a DMA bus master; it makes no difference.
- The output is a 23-bit *physical address* and a 2-bit *address type*. The address type specifies one of the four address spaces indicated at the right of the diagram.
- The four address spaces are to the right. The space corresponding to the incoming virtual address is a function of both the address and the memory type. Note that only the top two memory spaces (Multibus I/O and Multibus Memory) are accessible by way of the Multibus; the two On-Board memory spaces are accessed directly and are seldom of concern to non-Sun driver developers.

Programs can only reference driver address spaces in terms of virtual addresses which are then translated by the MMU into physical addresses within the appropriate physical address space.

#### Allocation of Multibus Memory

Here are some notes about the allocation of Multibus Memory resources in the Sun system.

No devices may be assigned addresses below 0x40000 in Multibus memory space since the CPU uses these addresses for DVMA. (See the end of this chapter for a discussion of DVMA).

The table on the next page shows a map of how Multibus Memory space is laid out in the Sun system. Note that this memory map, as well as all of those that follow, is only a general guide. To be sure that you are not installing a device at a location that will put it in conflict with existing devices, it's necessary to check the configuration of the specific systems into which it will be installed. The best way to do so is to check the local config file for the physical addresses of the devices installed within the bus of interest. This will probably give you enough information, but if you still think that there may be a conflict, and if you have a Sun source license, you can check the driver header files to determine the amount of space consumed on the bus by existing devices. With the exception of the Sky board, these devices can be rearranged. Also note the possibility that your machine will have devices attached to it, and taking up bus space, even though those devices do not appear in the config file. This possibility exists because the `xxmmmap()` system call can sometimes be used to drive a device without installing it in the formal sense — see the *Mapping Devices Without Device Drivers* section of the *Driver Development Topics* chapter for more details.



Table 2-2 *Sun-2 Multibus Memory Map*

| <i>Address</i>    | <i>Device</i>                                  |
|-------------------|------------------------------------------------|
| 0x00000 — 0x3FFFF | DVMA Space (256 Kilobytes)                     |
| 0x40000 — 0x7FFFF | Sun Ethernet Memory (#1) (256 Kilobytes)       |
| 0x80000 — 0x83800 | SCSI (#1) (16 Kilobytes)                       |
| 0x84000 — 0x87800 | SCSI (#2) (16 Kilobytes)                       |
| 0x88000 — 0x8B800 | Sun Ethernet Control Info (#1) (16 Kilobytes)  |
| 0x8C000 — 0x8F800 | Sun Ethernet Control Info (#2) (16 Kilobytes)  |
| 0x90000 — 0x9F800 | *** FREE *** (64 Kilobytes)                    |
| 0xA0000 — 0xAF800 | Sun Ethernet Memory (#2) (64 Kilobytes)        |
| 0xB0000 — 0xBF800 | *** FREE *** (64 Kilobytes)                    |
| 0xC0000 — 0xDF800 | Sun Model 100/150 Frame Buffer (128 Kilobytes) |
| 0xE0000 — 0xE1800 | 3COM Ethernet (#1)                             |
| 0xE2000 — 0xE3800 | 3COM Ethernet (#2)                             |
| 0xE4000 — 0xE7C00 | *** FREE *** (16 Kilobytes)                    |
| 0xE8000 — 0xF7800 | Reserved for Color Devices (64 Kilobytes)      |
| 0xF8000 — 0xFF800 | *** FREE *** (16 Kilobytes)                    |

### Allocation of Multibus I/O Space

Multibus I/O address space is specified in the config file as `mbio`. From the PROM monitor, Multibus I/O space begins at 0xEB0000, and extends to 0xEC0000.

Prior to Sun Release 3.0, the system made the assumption that any address lower than 0x10000 that it found in its config file was a Multibus I/O address. With current releases this is no longer true; now the bus type of every address must be explicitly given.

The following table of generic Multibus I/O usage, like the table above, is intended only as a guide.

Table 2-3 *Sun-2 Multibus I/O Map*

| <i>Address</i>  | <i>Device Type</i>                              |
|-----------------|-------------------------------------------------|
| 0x0040 — 0x0047 | Interphase Disk Controllers                     |
| 0x00A0 — 0x00A3 | CPC TapeMaster Controllers                      |
| 0x0200 — 0x020F | Archive Tape Drives                             |
| 0x0400 — 0x047F | Ikon 10071-5 Multibus/Versatec Interface        |
| 0x0480 — 0x057F | Systech VPC-2200 Versatec/Centronics Interfaces |
| 0x0620 — 0x069F | Systech MTI-800/1600 terminal Interface         |
| 0x2000 — 0x200F | Sky Board                                       |
| 0xEE40 — 0xEE4F | Xylogics 450/451 Disk Controller                |
| 0xEE60 — 0xEE6F | Xylogics 472 Multibus Tape Controller           |

## 2.2. VMEbus Machines

VMEbus machine architecture is generally more complex than Multibus machine architecture — it makes no distinction between I/O space and Memory space, but on the other hand it supports multiple address spaces. It does so for reasons of both cost and flexibility. The VMEbus was designed to be cost-effective for a range of applications. It is expensive (in terms of money, power, and board space) to provide the hardware for a full 32-bit address space. If installed devices only respond to 16-bit addresses, it makes sense to be able to put them all into a 16-bit address space and save the cost of 16-bits' worth of address decoders and the like. The 24 and 32-bit address spaces are similar compromises between cost and flexibility.

The driver writer has to understand which address space his board uses (generally, this is completely out of his/her control), and make an appropriate entry in the config file. For DMA devices, the driver writer has to know the address space that the board uses for its DMA transfers (this is usually a 32 or 24-bit space).

### Sun-2 VMEbus Address Spaces

The Sun-2 VMEbus machines are based upon the 24-bit subset of the generic VMEbus — they support only a 16-bit and a 24-bit address space. These address spaces are known as `vme16d16` (16 address bits and 16 data bits respectively) and `vme24d16` (24 address bits and 16 data bits). Sun-2 VMEbus machines also contain on-board memory and I/O space, of course, but these aren't accessed by way of the VMEbus and are only barely relevant to the driver developer.

There are four types of memory on Sun-2 VMEbus machines:

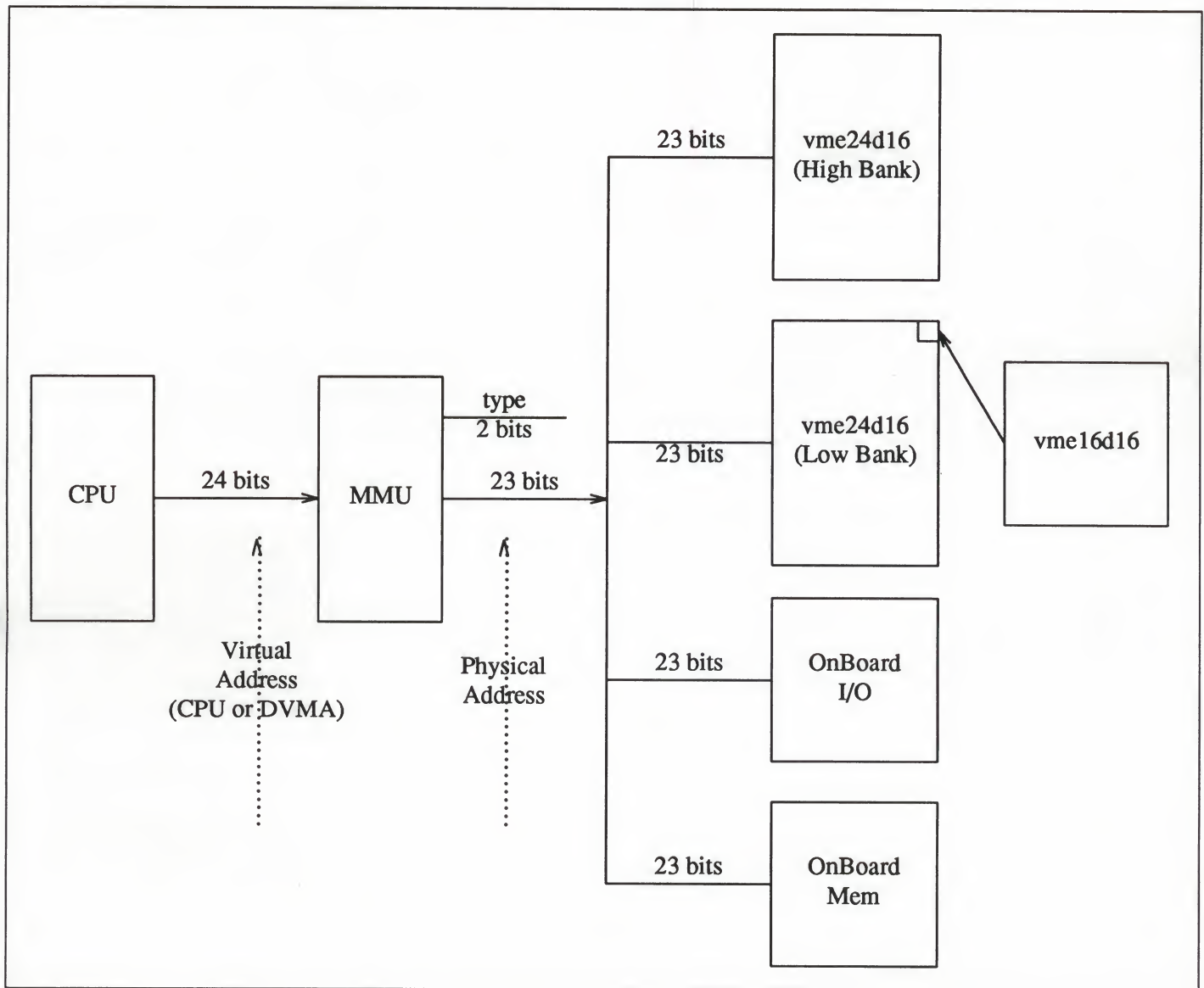
Table 2-4 *Sun-2 VMEbus Memory Types*

| <i>Description</i>                                                                   | <i>Address Size</i> | <i>Address Range</i> |
|--------------------------------------------------------------------------------------|---------------------|----------------------|
| On-Board Memory                                                                      | 23 bits             | 0x0 - 0x7FFFFFFF     |
| On-Board I/O Space                                                                   | 23 bits             | 0x0 - 0x7FFFFFFF     |
| <code>vme24d16</code>                                                                | 23+1 bits           | 0x0 - 0xFEFFFFF      |
| <code>vme16d16</code> — Stolen from top 64K of <code>vme24d16</code> (0x0 - 0xFFFFF) |                     |                      |

The four address spaces are laid out as follows:



Figure 2-2 Sun-2 VMEbus Address Spaces



Note a few details:

- In all Sun-2 machines (as in Sun-3's, Sun-3x's and Sun-4's), the address input into the MMU is a virtual address, and may originate with either the CPU or a DVMA (Direct Virtual Memory Access) bus master. (See the *Sun Main-Bus DVMA* section, later in this chapter, for a discussion of DVMA).
- Unlike Sun-2 Multibus systems, in which each memory type maps cleanly to one address space, vme24d16 maps to two different memory banks. Addresses from 0x0 to 0x7FFFFF are "type 2" memory, while those from 0x800000 and up are "type 3". This is because Sun-2 VMEbus machines have only 23 output address bits, and this trick is necessary to generate the full range of a 24-bit address space. (See *Sun-2 Address Mapping* in the

*Driver Development Topics* chapter for more details).

- Multibus boards, connected to VMEbus to Multibus adapters, can be plugged into physical memory anywhere within `vme24d16` (which means that they can also be in `vme16d16`).
- The 24 bits in the `vme24d16` address space are referred to in the above table as 23+1 bits. This is because, as should be clear in the diagram above, the Sun-2 MMU outputs only the lower 23 bits of the address, and the 24th bit is actually one of the MMU's type bits.
- Note especially that `vme16d16` is *stolen from* `vme24d16`. It's selected by addresses in the form `0xFFXXXX`, that is, addresses which have the 8 high bits set.

### Sun-3/Sun-3x/Sun-4 Address Spaces

Sun-3, Sun-3x and Sun-4 machines are all based on the full 32-bit VMEbus, so let's begin their discussion with a listing of the address types supported by the generic VMEbus. In all these memory references, we are referring to virtual VMEbus addresses, not Sun physical memory locations.

Table 2-5 *Generic VMEbus (Full Set)*

| <i>VMEbus-Space Name</i> | <i>Address Size</i> | <i>Data Transfer Size</i> | <i>Physical Address Range</i> |
|--------------------------|---------------------|---------------------------|-------------------------------|
| <code>vme32d16</code>    | 32 bits             | 16 bits                   | <code>0x0 - 0xFFFFFFFF</code> |
| <code>vme24d16</code>    | 24 bits             | 16 bits                   | <code>0x0 - 0xFFFFFFF</code>  |
| <code>vme16d16</code>    | 16 bits             | 16 bits                   | <code>0x0 - 0xFFFF</code>     |
| <code>vme32d32</code>    | 32 bits             | 32 bits                   | <code>0x0 - 0xFFFFFFFF</code> |
| <code>vme24d32</code>    | 24 bits             | 32 bits                   | <code>0x0 - 0xFFFFFFF</code>  |
| <code>vme16d32</code>    | 16 bits             | 32 bits                   | <code>0x0 - 0xFFFF</code>     |

Not all of these spaces are commonly used, but they are all nevertheless supported by the Sun-3 and Sun-4 lines. The following table indicates their sizes and physical address mappings.

Table 2-6 *Sun-3/Sun-4 VMEbus Address types*

| <i>Type</i> | <i>Address-Space Name</i>                                                                              | <i>Address Size</i> | <i>Address Range</i>           |
|-------------|--------------------------------------------------------------------------------------------------------|---------------------|--------------------------------|
| 0           | On-board Memory                                                                                        | 32 bits             | <code>0x0 - 0xFFFFFFFF</code>  |
| 1           | On-board I/O                                                                                           | 24 bits             | <code>0x0 - 0xFFFFFFF</code>   |
| 2           | <code>vme32d16</code>                                                                                  | 32 bits             | <code>0x0 - 0xFEFFFFFFF</code> |
| 3           | <code>vme32d32</code>                                                                                  | 32 bits             | <code>0x0 - 0xFEFFFFFFF</code> |
| 2           | <code>vme24d16</code> — Stolen from top 16M of <code>vme32d16</code> ( <code>0x0 - 0xFEFFFFFF</code> ) |                     |                                |
| 2           | <code>vme16d16</code> — Stolen from top 64K of <code>vme24d16</code> ( <code>0x0 - 0xFFFF</code> )     |                     |                                |
| 3           | <code>vme24d32</code> — Stolen from top 16M of <code>vme32d32</code> ( <code>0x0 - 0xFEFFFFFF</code> ) |                     |                                |
| 3           | <code>vme16d32</code> — Stolen from top 64K of <code>vme24d32</code> ( <code>0x0 - 0xFFFF</code> )     |                     |                                |

The Sun-3x is different than the Sun-3 and Sun-4 in that the hardware does not use page table entries (PTE's) with a type identifier to map the devices into physical memory. The Sun-3x uses absolute physical address when mapping devices.



Therefore the type field is not used as an identifier of physical address mapping. The next two tables show the virtual VME addresses and the corresponding physical addresses for the specific ranges. Note for the Sun-3x there is no vme32d16 entry and there is a hole in the address space usage from the end of the on-board I/O area to the beginning of the vme16d16 area.

Table 2-7 *Sun-3x VMEbus Address types*

| Type | Address-Space Name                                          | Address Size | Address Range    |
|------|-------------------------------------------------------------|--------------|------------------|
| 0    | On-board Memory                                             | 32 bits      | 0x0 — 0xFFFFFFFF |
| 1    | On-board I/O                                                | 24 bits      | 0x0 — 0x00FFFFFF |
| 2    | vme24d16                                                    | 32 bits      | 0x0 — 0xFEFFFFFF |
| 3    | vme32d32                                                    | 32 bits      | 0x0 — 0xFEFFFFFF |
| 2    | vme16d16 — Stolen from top 64K of vme24d16 (0x0 - 0xFFFF)   |              |                  |
| 3    | vme24d32 — Stolen from top 16M of vme32d32 (0x0 - 0xFEFFFF) |              |                  |
| 3    | vme16d32 — Stolen from top 64K of vme24d32 (0x0 - 0xFFFF)   |              |                  |

Table 2-8 *Sun-3x Physical Address map*

| Type <sup>†</sup> | Address-Space Name | Address Size | Address Range           |
|-------------------|--------------------|--------------|-------------------------|
|                   | On-board Memory    | 32 bits      | 0x00000000 — 0x57FFFFFF |
|                   | On-board I/O       | 32 bits      | 0x58000000 — 0x6EFFFFFF |
|                   | vme16d16           | 32 bits      | 0x7C000000 — 0x7C00FFFF |
|                   | vme16d32           | 32 bits      | 0x7D000000 — 0x7D00FFFF |
|                   | vme24d16           | 32 bits      | 0x7E000000 — 0x7EFFFFFF |
|                   | vme24d32           | 32 bits      | 0x7F000000 — 0x7FFFFFFF |
|                   | vme32d32           | 32 bits      | 0x80000000 — 0xFFFFFFFF |

<sup>†</sup>Types are not used with the Sun-3x architecture.

Sun-3/Sun-3x/Sun-4 space overlays are much more complex than those of the Sun-2, as is evident from both the tables above and the diagrams below. The principle, however, is the same — when a space overlays a larger space, its memory is stolen from that larger space and is considered by the MMU to be in the overlaid space. One simply cannot address above 0xFF000000 in 32-bit VMEbus space or above 0x00FF0000 in 24-bit VMEbus space.

As the following diagrams illustrate, Sun-3 and Sun-4 addressing schemes are almost identical. They differ only in the size of the virtual address which — output by the CPU or a DVMA Bus Master — is fed to the MMU.

The Sun-3x, which has the MMU on the CPU chip, is a different hardware architecture than the Sun-3's and Sun-4's. There is a full 32 bit input to the MMU from the CPU, and all 32 bits are used for input to the OnBoard and vme modules. No devices use the vme32d16 so it is not part of the memory map.

Figure 2-3 Sun-3 VMEbus Address Spaces

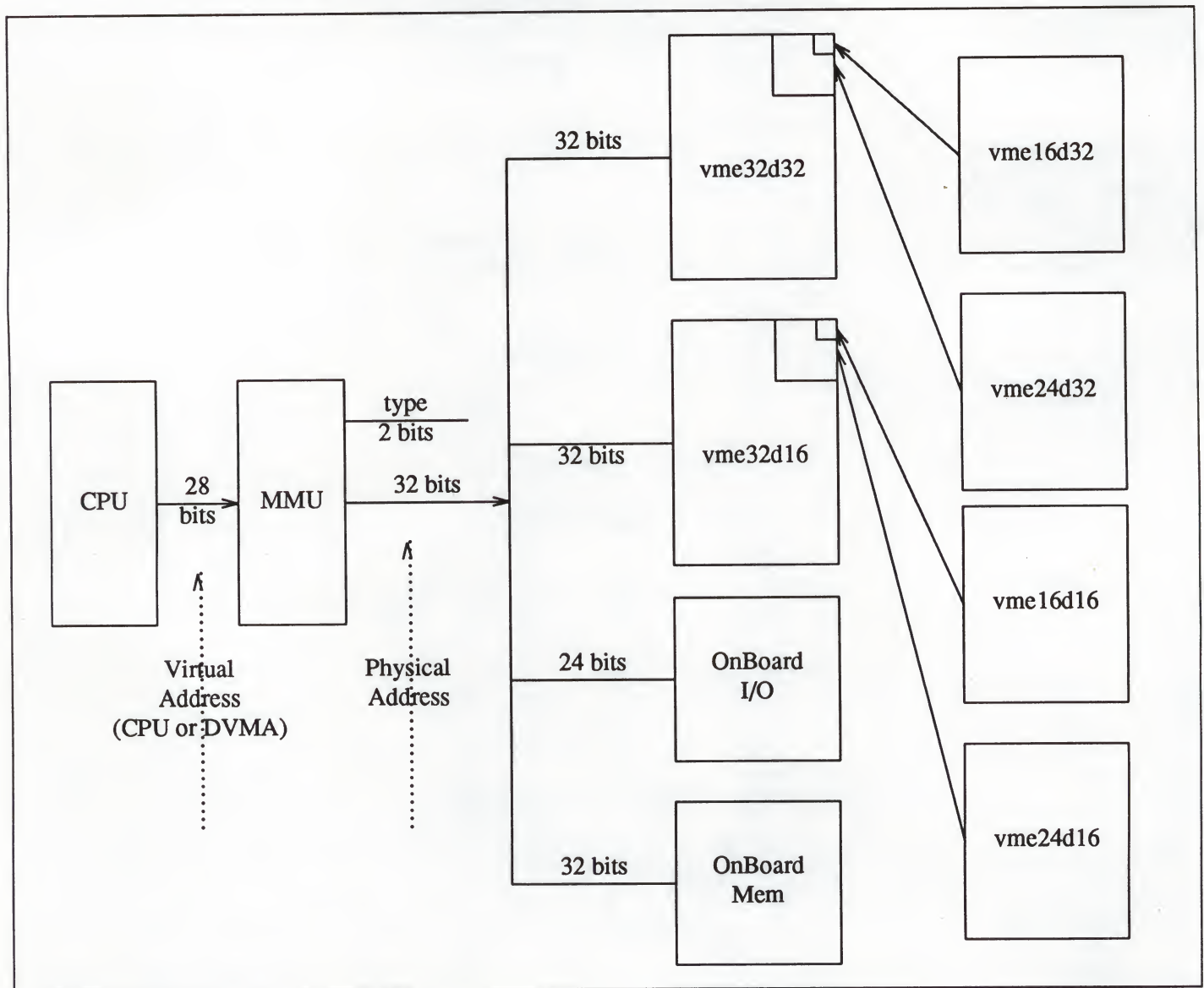




Figure 2-4 Sun-3x VMEbus Address Spaces

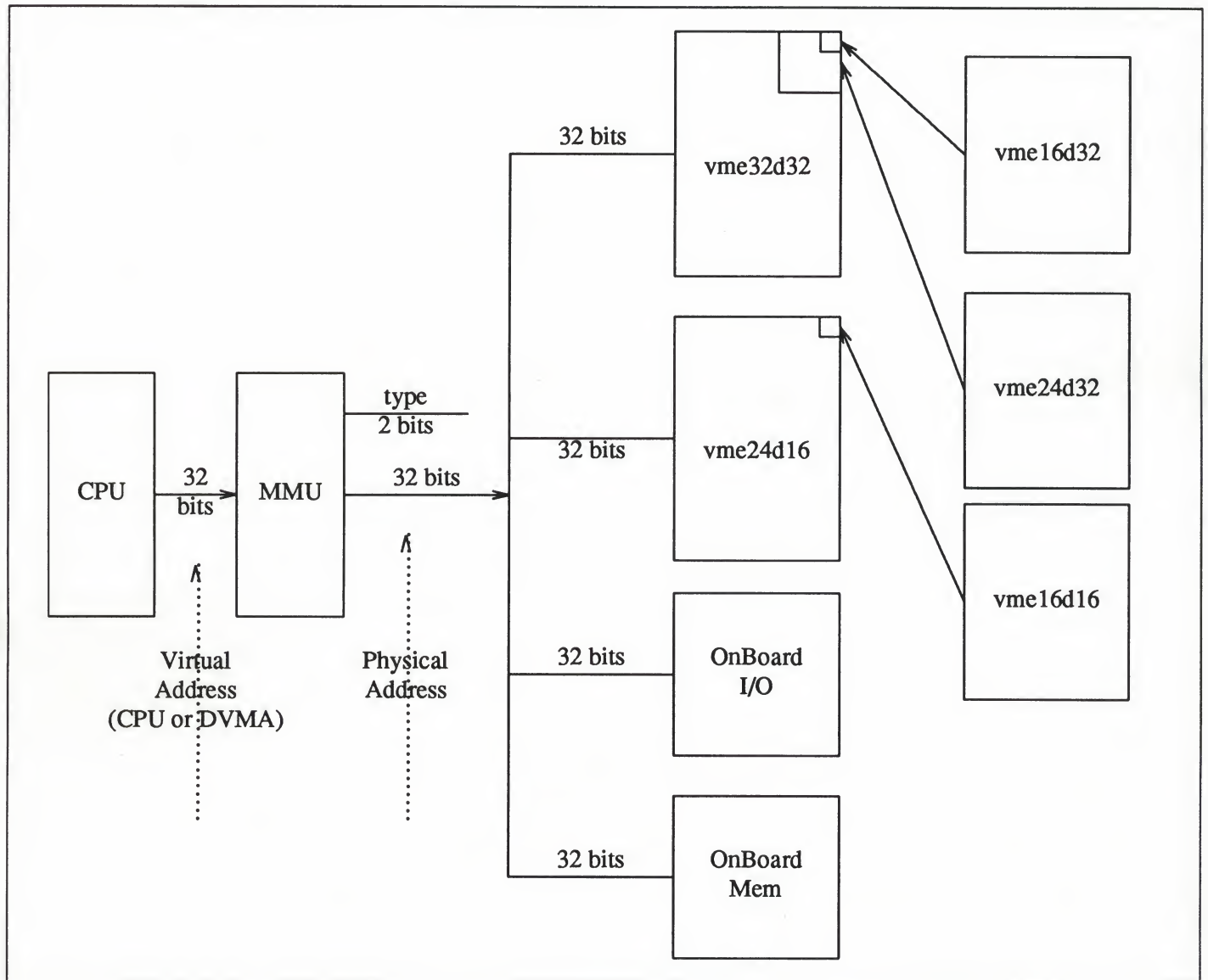
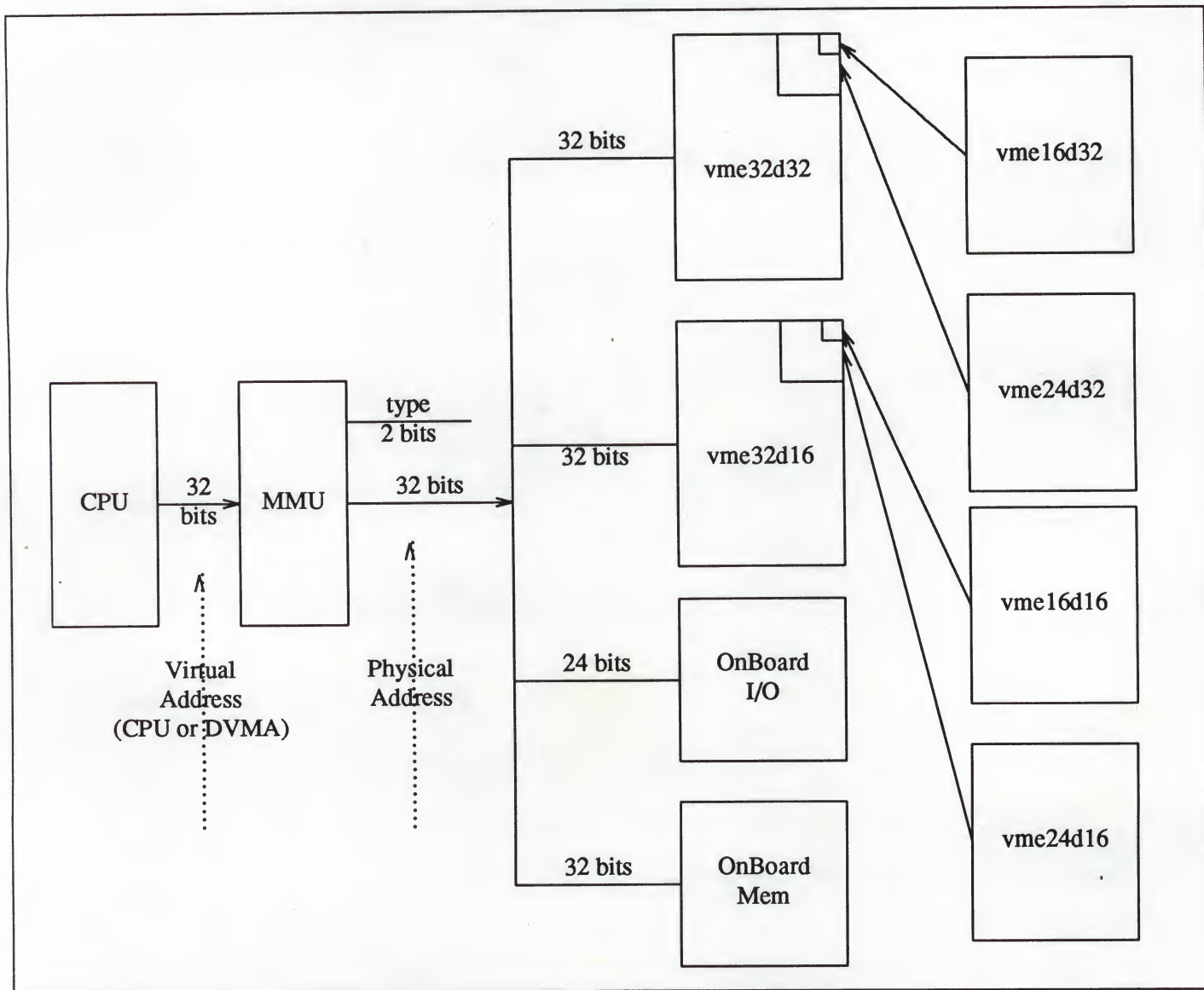


Figure 2-5 Sun-4 VMEbus Address Spaces



### Allocation of VMEbus Memory

This section summarizes the typical use of the 16, 24 and 32-bit VMEbus address spaces by Sun devices. Note that the usages summarized here are only for the generic configuration, and there's no guarantee that they match the exact usage on your machine. They will, however, help you to decide where to attach your device. The "Allocated From" field shows whether bus space is allocated from the high end of the given range or from the low end. The idea is to keep the maximum size "hole" in the middle in case the boundary needs to be shifted later.

Table 2-9 16-bit VMEbus Address Space Allocation

| <i>Address Range</i> | <i>Allocated From</i> | <i>Description of Use</i>     |
|----------------------|-----------------------|-------------------------------|
| 0x0000-0x7FFF        | Low                   | Reserved for OEM/user devices |
| 0x8000-0xFFFF        | High                  | Reserved for Sun devices      |

16-bit VMEbus space is mapped into the topmost 64K of 24-bit VMEbus space at 0x00FF0000 to 0x00FFFFFF (on Sun-2s) or 0xFFFF0000 to 0xFFFFFFFF (on Sun-3's, Sun-3x's, and Sun-4's). Note: The Multibus/VMEbus Adapter will map the Multibus I/O addresses of Multibus cards that use Multibus I/O into the same addresses in the 16-bit VMEbus space. This may place the standard Multibus addresses for some cards into the OEM/user area in the above table. These addresses can be changed, if necessary, by physically readdressing the device and then changing its entry in the config file.

Table 2-10 24-bit VMEbus Address Space Allocation

| <i>Address Range</i> | <i>Allocated From</i> | <i>Description of Use</i>           |
|----------------------|-----------------------|-------------------------------------|
| 0x000000-0x0FFFFF    | Low                   | CPU board DVMA space                |
| 0x100000-0x1FFFFF    |                       | Reserved by Sun                     |
| 0x200000-0x2FFFFF    |                       | Reserved for small Sun devices      |
| 0x300000-0x3FFFFF    |                       | Reserved for large Sun devices      |
| 0x400000-0x7FFFFF    | (Taken)               | Reserved for huge Sun devices       |
| 0x800000-0xBFFFFF    | High                  | Reserved for huge OEM/user devices  |
| 0xC00000-0xCFFFFF    | Low                   | Reserved for large OEM/user devices |
| 0xD00000-0xDFFFFF    | High                  | Reserved for small OEM/user devices |
| 0xE00000-0xEFFFFF    |                       | Multibus-to-VMEbus memory space     |
| 0xF00000-0xFEFFFF    |                       | Reserved for the Future             |
| 0xFF0000-0xFFFFFFFF  |                       | Stolen by 16-bit VMEbus space       |

Table 2-11 32-bit VMEbus Address Space Allocation (Sun-3's, Sun-3x's, and Sun-4's)

| <i>Address Range</i>    | <i>Description of Use</i>     |
|-------------------------|-------------------------------|
| 0x00000000 - 0x000FFFFF | DVMA Space                    |
| 0x00100000 - 0x7FFFFFFF | Reserved by Sun               |
| 0x80000000 - 0xFEFFFFFF | Reserved for OEM/user devices |
| 0xFF000000 - 0xFFFFFFFF | Stolen by vme24d16            |



These same assignments apply to both 16-bit-data and 32-bit-data VMEbus accesses. Note that, at least in the GENERIC kernel, there are some Sun devices (tm0, tm1, vpc0, vpc1 and mti0-4) installed in the OEM/user area. It's always best to check, when choosing an installation address, that you aren't going to conflict with an already installed device.

Table 2-12 VMEbus Address Assignments for Some Devices

| <i>Device</i>      | <i>Addressing</i> | <i>Addresses Used</i>            |
|--------------------|-------------------|----------------------------------|
| VMEbus SKY Board   | vme16d16          | 0x8000 - 0x8FFF (Sun-2 only)     |
| VMEbus SCSI Board  | vme24d16          | 0x200000 - 0x2007FF              |
| VMEbus TOD Chip    | vme24d16          | 0x200800 - 0x2008FF (Sun-2 only) |
| Graphics Processor | vme24d16          | 0x210000 - 0x210FFF              |
| Sun-2 Color Board  | vme24d16          | 0x400000 - 0x4FF7FF              |

The VMEbus Sky board occupies addresses 8000-8FFF in 16-bit address space, and it requires that the high nibble of the address be '8'. Unlike other pre-installed devices, it cannot be moved.

This table is, of course, not complete. There is always a variety of devices on the bus, as can be easily determined by examining the config file. This table, however, does include the standard devices that use a significant amount of space on the VMEbus. Note that, in machines which came after the Sun-2 line, several of these devices have been replaced by on-board devices and have thus disappeared from the VMEbus address space.

#### The Sun VMEbus to Multibus Adapter

Multibus devices that are to be attached to VMEbus machines must be attached to a VMEbus to Multibus adapter. (The Adapter works for most, but not all, Multibus boards). An adapter can be used to take over *one and only one* chunk of vme24d16. However, that chunk can overlap all or part of vme16d16 (because vme16d16 is a proper subset of vme24d16). In any case, the adapter must be told how much space the board attached to it actually expects, for by default it will take over a full megabyte. Note that the Multibus Adapter supports fully vectored interrupts, and that drivers for Multibus devices attached by way of adapters need not poll, since the adapters contain switches by which Multibus devices can be assigned vectors.

#### Interrupt Vector Assignments

The table below shows the assignments of interrupt vectors for those devices that can supply interrupts through the VMEbus vectored interrupt interface. To pick one for your device, examine the kernel config file for an unused number in the range reserved for customer use, 0xC8 to 0xFF.



Table 2-13 *Vectored Interrupt Assignments*

| <i>Vector Numbers</i> | <i>Description</i>                                |
|-----------------------|---------------------------------------------------|
| 0x40 thru 0x43        | sc0, sc?, si0, si? — SCSI Host Adapters           |
| 0x48 thru 0x4B        | xyz0, xyz1, xyz? — Xylogics Disk Controllers      |
| 0x4C thru 0x5F        | future disk controllers                           |
| 0x60 thru 0x63        | tm0, tm1, tm? — TapeMaster Tape Controllers       |
| 0x64 thru 0x67        | xtc0, xtc1, xtc? — Xylogics Tape Controllers      |
| 0x68 thru 0x6F        | future tape controllers                           |
| 0x70 thru 0x73        | ec? — 3COM Ethernet Controller                    |
| 0x74 thru 0x77        | ie0, ie1, ie? — Sun Ethernet Controller           |
| 0x78 thru 0x7F        | future ethernet devices                           |
| 0x80 thru 0x83        | vpc? — Systech VPC-2200                           |
| 0x84 thru 0x87        | vp? — Ikon Versatec Parallel Interface            |
| 0x88 thru 0x8B        | mti0, mti? — Systech Serial Multiplexors          |
| 0x8C thru 0x8F        | dcp1, dcp? — SunLink Comm. Processor              |
| 0x90 thru 0x9F        | zs0, zs1 — Sun-3/Sun-3x Terminal/Modem Controller |
| 0xA0 thru 0xA3        | future serial devices                             |
| 0xA4 thru 0xA7        | pc0, pc1, pc2, pc3 — SunIPC                       |
| 0xA8 thru 0xAB        | future frame buffer devices                       |
| 0xAC thru 0xAF        | future graphics processors                        |
| 0xB0 thru 0xB3        | sky0, ? — SKY Floating Point Board                |
| 0xB4 thru 0xB7        | SunLink Channel Attach                            |
| 0xB8 thru 0xC7        | Reserved for Sun Use                              |
| 0xC8 thru 0xFF        | Reserved for Customer Use                         |

### 2.3. ATbus Machines

The Intel 80386 processor handles I/O devices placed in either memory space or in I/O space. On the 80386, memory-mapped I/O provides additional programming flexibility. Any memory instruction can access any I/O port located in the memory space. For example, the MOV instruction transfers data between any register and any port. The AND, OR, and TEST instructions manipulate bits in the internal registers of a device.

On some devices, reading a register will not read back what was written. Therefore, instructions such as AND, OR, and TEST can, in some cases, produce unexpected results because the instruction reads a good location, changes it, and writes it back. See the *Other Device Peculiarities* section, ahead.

Memory-mapped I/O can use the full complement of instructions. The 16 MB memory of AT memory exists in the 4 GB physical address space of the Sun386i at 0xE000 0000. For example, a device that, on an AT, shows up in memory at D0 0000 will show up in the Sun386i physical memory at 0xE0D0 0000. Virtual addresses are assigned during the autoconfiguration process.

If an I/O device is mapped into the I/O space then the IN, OUT, INS, and OUTS instructions are used to communicate to and from the device. All I/O transfers

are performed via the AL (8-bit), AX (16-bit), or EAX (32-bit) registers. The first 256 bytes of the I/O space are directly addressable. The entire 64 Kbyte I/O space is indirectly addressable through the DX register.

The Sun386i has 21 interrupt channels, but only 11 are available to devices on the AT bus. The following list of interrupt channel assignments shows all of the interrupt channels.

Table 2-14 *Interrupt Channel Assignments*

| <i>AT Channel*</i>      | <i>Assignee</i>                 |
|-------------------------|---------------------------------|
| 3                       | AT Pin B25                      |
| 4                       | AT Pin B24                      |
| 5                       | AT Pin B23                      |
| 6                       | Not available (system diskette) |
| 7                       | Not available (parallel port)   |
| 8                       | SCSI                            |
| 9                       | AT Pin B04                      |
| 10                      | AT Pin D03                      |
| 11                      | AT Pin D04                      |
| 12                      | AT Pin D05                      |
| 13                      | Not available (Ethernet)        |
| 14                      | AT Pin D07                      |
| 15                      | AT Pin D06                      |
| * Available to AT Cards |                                 |

When you add an AT card to the AT bus, you must select one of the values in the Channel column for the AT card's jumpers. For example, if you select channel 10 for a serial card, the "device" line in the config file might look as follows:

```
device ns0 at atio ? csr 0x3f8 irq 10 priority 6
```

The Sun386i does not permit two AT cards to use the same interrupt channel.

Some cards will also use DMA and will have jumpers to select a DMA channel to use. The following list shows that DMA channels 0-3 and channel 5 are available for AT cards. Note that channel 0 and 5 can be used with 16-bit DMA devices; 1, 2, and 3 can be used only with 8-bit DMA devices. Note also that channels 4, 6, and 7 are pre-assigned.



Table 2-15 *Sun386i DMA Channel Assignments*

| <i>Channel</i> | <i>Assignee</i> | <i>Size (bits)</i> |
|----------------|-----------------|--------------------|
| 0              | AT Bus          | 16                 |
| 1              | AT Bus          | 8                  |
| 2              | AT Bus          | 8                  |
| 3              | AT Bus          | 8                  |
| 4              | Software        | Not Available      |
| 5              | AT Bus          | 16                 |
| 6              | Ethernet        | 16                 |
| 7              | SCSI            | 16                 |

For example, you might set up a controller that uses DMA channel 3. For this, the “controller” line in the config file might look like: this:

```
controller wds0 at atio ? csr 0x320 dmachan 3 irq 3 priority 3
```

The Sun386i does not permit two AT cards to use the same DMA channel.

In these examples, “priority” refers to the spl levels used in the driver. That is, the phrase “priority 3” implies that the driver uses spl3() to protect its critical regions.

## Loadable Drivers

On Sun386i machines, device drivers can be dynamically loadable. That is, they can be attached to a system without rebuilding its kernel and without having to bring the system down and restart it. See the *Adding and Removing Loadable Drivers* section of the *Configuring the Kernel* chapter for details.

## DOS and SunOS Environments

The Sun386i system supports both DOS drivers and SunOS drivers.

You can attach a DOS device driver in the standard DOS way, but it will be usable only from within the DOS environment. Usually, all you need to do is to first plug in an add-in board. Then you insert an installation diskette (which comes with the board) into Drive A> and re-boot the system. The device driver is already compiled and linked. Generally, the diskette contains programs called “INSTALL” or something similar. You execute this program by typing its name. It copies the driver file from the diskette to the hard disk. At the same time, this procedure will modify the disk’s config.sys file.

The DOS system must be re-booted. The device driver will automatically be loaded into memory, its options will be parsed, and the driver will be initialized.

**NOTE** *The DOS driver on the Sun386i is running under SunOS and DOS, but the driver is unaware of this. SunOS might switch control to another task during device operation, so strict timing dependencies could fail. Real time devices, for example, may not work properly. If a peripheral and controller have strict timing requirements, their drivers should be written in the standard SunOS style. DOS drivers do not run at the elevated priority of SunOS drivers.*



SunOS drivers, of course, are parts of the system kernel. Thus the timing requirements of most devices can be met under SunOS. SunOS drivers are accessible from the DOS environment.

## 2.4. Hardware Peculiarities to Watch Out For

There is a variety of device peculiarities that the driver developer must be aware of. The most common of them are related to the Multibus and Multibus-based devices, but there are others as well.

### Multibus Device Peculiarities

The IEEE Multibus is a source of problems for two separate reasons. The first of these, discussed immediately below, is the fact that the Multibus has a different notion of byte order than does the either Motorola MC680X0 family or the Sun SPARC processor (the reduced instruction set CPU upon which Sun-4 machines are built). The second is simply that the Multibus has been around for a long time, and thus brings with it a variety of older devices, many of which have addressing limitations and other characteristics which make for a less than perfect fit with the Sun architecture.

### Multibus Byte-Ordering Issues

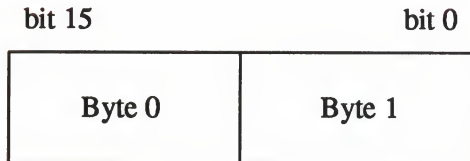
The Sun-2, Sun-3, and Sun-3x processors are members of the Motorola MC680X0 family, while Sun-4 processors are based on the SPARC CPU. All of these processors address bytes within words by what we shall call *IBM conventions* — the most significant byte of a word is stored at the lowest addressed byte of the word. The Multibus, on the other hand, uses *DEC conventions* — the least significant byte of a word is stored at the lowest address, and significance increases with address.

This class of byte-addressing conventions leads to two separate problems, with two separate solutions:

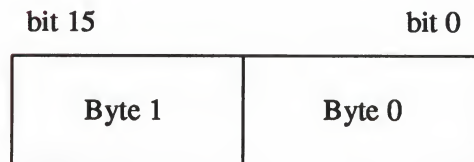
- The first problem occurs when you're moving a single *byte* across the interface between the MC680X0/SPARC and the IEEE Multibus. Because the two devices don't agree about the end of the word that the byte actually appears in, you have to change the byte address before the move — what you want to do, in effect, is move every byte to the other side of the word which it occupies — the most CPU-efficient way of doing so is to toggle the least significant bit of every byte address.
- The second problem, also related to the Multibus, is a higher level version of the first. It occurs when machine *words* with significant internal structure (or structures that contain words) are moved across the bus interface. (If you write only words, and the device uses only words, there's no problem). The Multibus byte-ordering incompatibility will cause structures to be scrambled when they're moved across the bus interface, unless the bytes within them are physically swapped first.

Here are a few pictures describing the problems in detail:

## Motorola (IBM) Byte Ordering



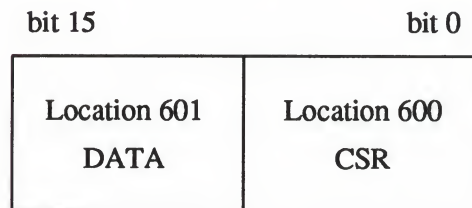
## Multibus (DEC) Byte Ordering



That is, the MC680X0 and SPARC CPUs place byte 0 in bits 8 through 15 of the 16-bit word, whereas the Multibus places byte 1 in those bits. If you did everything with the CPU, or everything on the Multibus, there wouldn't be any conflict, since things would be consistent. However, as soon as you cross the boundary between them, the byte order is reversed. Thus, you have to toggle the least significant bit of the address of any *byte* destined for the Multibus — this will have the effect of swapping adjacent addresses and thus reordering the bytes.

To clarify this, consider an interface for a hypothetical Multibus board containing only two 8-bit I/O registers, namely a control and status register (csr) and a data register (we actually use this design later on in our example of a simple device driver). In this board, we place the command and status register at Multibus byte location 600, and the data register at Multibus byte location 601. The Multibus picture of that device looks like this:

## Hypothetical Board Registers



But the MC680X0 and SPARC processors view that device as looking like this:



## Hypothetical Board Registers

| bit 15              | bit 0                |
|---------------------|----------------------|
| Location 600<br>CSR | Location 601<br>DATA |

so that if you were to read location 600 from the point of view of the processor, you'd really end up reading the DATA register off the Multibus instead. So, when we define the *skdevice* data structure for that board, we define it by starting with the register definition in the device manual, and then swapping bytes to take account of the expected byte swapping:

```
struct skdevice {
    char    sk_data;    /* 01: Data Register */
    char    sk_csr;     /* 00: command(w) and status(r) */
};
```

This rule (flipping the least significant bit of the address) holds good for all *byte* transfers which cross the line between the MC680X0/SPARC CPU and the Multibus.

## Other Multibus-related Peculiarities

- Many Multibus device controllers are geared for the 8-bit 8080 and Z80 style chips and don't understand 16-bit data transfers. Because of this, such controllers are quite happy to place what's really a word quantity (such as a 16-bit address which must be two-byte aligned in the MC680X0) starting on an odd byte boundary. Some devices use 16-bit or 20-bit addresses (many don't know about 24-bit addresses), and it often happens that you have to chop an address into bytes by shifting and masking, and assign the halves or thirds of the address one at a time, because the device controller wants to place word-aligned quantities on odd-byte boundaries. Note also that many Multibus boards are geared for the 8086 family with its segmented address scheme. An 8086 (20-bit) address really consists of a 4-bit segment number and a 16-bit address; you usually have to deal with the 4-bit part and the 16-bit part separately. For a good example of what we're talking about here, see the code for *vp.c* in the *Sample Driver Listings* appendix to this manual.
- Although there are a myriad of vendors offering Multibus products, remember that the Multibus is a "standard" that evolved from a bus for 8-bit systems to a bus for 16-bit systems. Read vendors' product literature *carefully* (especially the fine print) when selecting a Multibus board. The memory address space of the Multibus is *supposed to be* 20 or 24 bits wide and the I/O address space of the Multibus is *supposed to be* 16 bits wide. In practice, some older boards are limited to 16 bits of address space and 8 bits



of I/O space. In particular, watch for the following addressing peculiarities:

- For a memory-mapped board, ensure that the board can actually handle a full twenty bits of addressing. Older Multibus boards often can only handle sixteen address lines. The Sun system assumes there is a 20-bit Multibus memory space out there. If the Multibus board you're talking to can only handle 16-bit addresses, it will ignore the upper four address lines, and this means that such a board "wraps around" every 64K, which means that on a Sun the addresses that such a board responds to would be replicated sixteen times through the one-megabyte address space on the Multibus. This may conflict with some other device.
- Some Sun-2 Multibus systems, notably Sun-2/170s, have a backplane structure that complicates the installation of 24-bit memory-mapped devices. The internal "bus" on these systems (often called the P2 bus) is divided into multiple segments, each mapped to a portion of the backplane slots. In such systems, 24-bit memory-mapped devices must be installed in a different segment than that used by standard Sun-2 devices. See the *Sun-2/170 Configuration Guide* for more information.
- For an I/O-mapped board (one that uses I/O registers), make sure that the board can handle 16-bit I/O addressing. Some older boards support only 8-bit I/O addressing. In our system, the address spaces of such boards would find themselves replicated every 256 bytes in the I/O address space. Trying to fit such a board into the Sun system would severely curtail the number of I/O addresses available in the system.
- Finally, watch out for boards containing PROM code that expects to find a CPU bus master with an Intel 8080, 8085, or 8086 on it. Such boards are of course useless in the Sun system.

### Sun-4/SPARC Peculiarities

There are two peculiarities which are specific to machines built upon the Sun SPARC CPU (currently, just Sun-4's) which can impact device drivers. For more information about the Sun-4 machine architecture, see *Porting C, Fortran and Pascal Programs to the Sun-4*.

- The first problem is structure alignment. In MC680X0 family processors, structures are aligned on half-word boundaries, but on Sun-4's, the structure-alignment requirements are imposed by the most strictly-aligned structure components. For example, a structure containing only bytes and characters has no alignment restrictions, while a structure containing a double word must be constructed so as to guarantee that that this word falls on a 64-bit boundary.

Programmers must be aware of these rules when writing drivers, for Sun-4 compilers will pad structures to enforce them, and such padding will not always be correct for structures intended to map to device registers. Also, structures must be carefully designed if drivers are to be portable across machine architectures.

- The second problem is data alignment. In MC680X0 family processors, characters are aligned on byte boundaries, while integers of all sizes are



aligned on 16-bit boundaries. In Sun-4 machines, in contrast, all quantities must be aligned on their “natural” boundaries: 16-bit half words on 16-bit boundaries, 32-bit words on 32-bit boundaries and 64-bit double words on 64-bit boundaries.

In normal programs, details such as these are handled by the compiler. In drivers, however, more care must be taken. SPARC (unlike the MC68010) doesn’t break down 32-bit transactions into successive 16-bit transactions. Thus, there are times when 32-bit entities have to be broken down by the driver if they are to get across the bus correctly. More specifically, 32-bit or 64-bit alignment is not possible in the 16-bit VMEbus spaces, and thus 32-bit and 64-bit data access does not exist. In the 32-bit VMEbus spaces, all data paths exist.

## Other Device Peculiarities

There are other device peculiarities of interest to the driver developer. These peculiarities are particularly unfortunate in that they tend to require special handling of various kinds — byte swapping, bit shuffling, timing delays, etc. — whenever the driver contacts the device. Such special handling precludes the most obvious and desirable means of interfacing the driver to the device, by mapping the device registers into a C-structure declaration and then accessing them by way of references to structure fields.

- One of the most infuriating of these peculiarities is internal sequencing logic. Devices with this strange characteristic (a vestige of microcomputer systems with extremely limited address space) map multiple internal registers to the same externally addressable address. There are various kinds of internal sequencing logic:
  - The Intel 8251A and the Signetics 2651 alternate the same external register between *two* internal mode registers. Thus, if you want to put something in the first mode register of an 8251, you do so by writing to the external register. This write will, however, have the invisible side effect of setting up the sequencing logic in the chip so that the next read/write operation refers to the alternate, or second, internal register.
  - The NEC PD7201 PCC has multiple internal data registers. To write a byte into one of them, it’s necessary to first load the first (register 0) with the number of the register into which the following byte of data will go — you then send that byte of data and it goes into the specified data register. The sequencing logic then automatically sets up the chip so that the next byte sent will go into data-register 0.
  - Another chip of a similar ilk is the AMD 9513 timer. This chip has a data pointer register for pointing at the data register into which a data byte will go. When you send a byte to the data register, the pointer gets incremented. The design of the chip is such that you *can’t read the pointer register to find out what’s in it!*
- In fact, it’s often true that device registers, when read, don’t contain the same bits that were last written into them. This means that bitwise operations (like `register &= ~XX_ENABLE`) that have the side effect of



generating register reads must be done in a software copy of the device register, and then written to the real device register.

- Another problem is timing. Many chips specify that they can only be accessed every so often. The Zilog Z8530 SCC, which has a “write recovery time” of 1.6 microseconds, is an example. This means that a delay has to be enforced (with DELAY) when writing out characters with an 8530. Things can get worse, however, for there are instances when it’s unclear what delays are needed, and in such cases it’s left to the driver developer to determine them empirically.
- And peripheral devices can contain chips that use a byte-ordering convention different from that used by the Sun system into which they’re installed. The Intel 82586, for example, supports DEC byte-ordering conventions; this makes it perfectly compatible with Multibus-based, but not VMEbus-based, Sun machines. Drivers for such peripheral devices will have to swap bytes, as indicated above, and to take care that, in doing so, they don’t inadvertently reorder the bits in any control fields greater than 16 bits in length.
- Finally, there are some common interrupt-related peculiarities worth noting:
  - When a controller interrupts, it does *not* necessarily mean that both it *and* one of its slave devices are ready. Some controllers are designed in this way, but others interrupt to indicate that the controller or one of its devices *but not necessarily both* is ready.
  - Not all devices power up with interrupts disabled and then start interrupting only when told to do so.
  - While there should be a way to determine that a board has actually generated an interrupt — an attention bit or something equivalent — some devices have no such thing.
  - Finally, an interrupting board should shut off its interrupts when told to do so (and also after a bus reset). Not all do.

## 2.5. DMA Devices

Many device controller boards are capable of what is known as Direct Memory Access or DMA. This means that the CPU can tell the device controller for such devices the address in memory where a data transfer is to take place and the length of the data transfer, and then instruct the device controller to start the transfer. The data transfer then takes place without further intervention on the part of the processor. When it’s complete, the device controller interrupts to say that the transfer is done.

### Sun Main-Bus DVMA

**NOTE** *Sun-2, Sun-3, Sun-3x, and Sun-4 machines use Direct Virtual Memory Access (DVMA) to allow devices on the Main Bus (either a VMEbus or a Multibus) to perform DMA transfers from and to system virtual address space. In the Sun386i system, however, the Memory Management Unit (MMU) is incorporated directly on the Intel 80386 chip itself; devices need to use physical addresses. Sun386i*



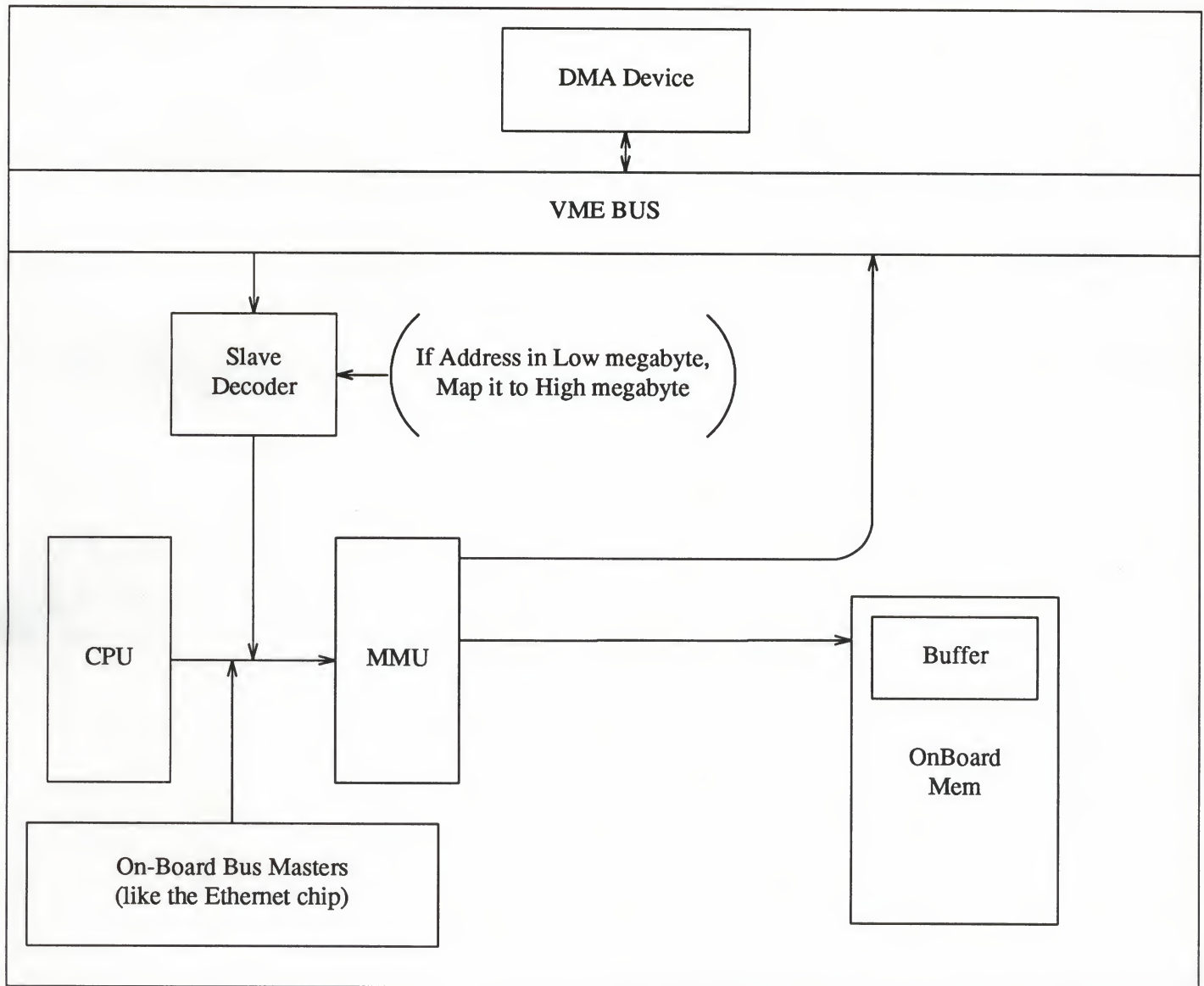
*DMA is discussed in the next Section.*

Direct Virtual Memory Access (DVMA) is a mechanism provided by the Sun Memory Management Unit to allow devices on the Main Bus (either a VMEbus or a Multibus) to perform DMA directly to Sun processor memory. It also allows Main Bus master devices to do DMA directly to Main Bus slaves without the extra step of going through processor memory. DVMA works by ensuring that the addresses used by devices are processed by the MMU, just as if they were virtual addresses generated by the CPU. This allows the system to provide the same memory protection and mapping facilities to DMA devices as it does to the system CPU (and thus to programs).

When setting up a driver to support DMA, it's necessary to know the device's DMA address size. This address size is the primary factor used in determining which of the system address spaces will host the device. Multibus devices generally have a DMA address size of 20 bits, while VMEbus devices generally have a 24 or 32-bit DMA address size.

- Since, on Sun-2 Multibus machines, DMA addresses are generally 20-bits long, the system DVMA hardware responds to the first 256K of Multibus address space (0x0 to 0x3FFFF). When an address in this range appears on the bus, the DVMA hardware adds 0xF00000 to it (the system places the Multibus memory address space at 0xF00000 in the system's virtual address space) and then uses the MMU to map to the location in physical memory that will be used for the data transfer.
- On Sun-2 VMEbus systems, the DVMA hardware responds to the entire lower megabyte of VMEbus address space (0x0 to 0xFFFFF). The system maps addresses in this range into the most significant megabyte of system virtual address space (0xF00000 to 0xFFFFFFF).
- On the Sun-3, Sun-3x, and Sun-4 systems the DVMA hardware responds to the lowest megabyte of VMEbus address space *in both the 24-bit and 32-bit VMEbus spaces*. It maps addresses in this megabyte into the most significant megabyte of system virtual address space (0xFF00000 to 0xFFFFFFFF for the Sun-3 and 0xFFF00000 to 0xFFFFFFFF for the Sun-3x and Sun-4). The Sun-3, Sun-3x, and Sun-4 DVMA hardware use supervisor access for checking protection.

The driver writer must account for these mappings, as should be evident from the diagram below.

Figure 2-6 *System DVMA*

Devices can only make DVMA transfers in memory buffers which are from (or redundantly mapped into — see below) the low-memory areas reserved as DVMA space. The memory-management hardware will then recognize references to these areas and map them into the high megabyte of system virtual address space, an area known as DVMA space. Likewise, if a driver needs to allocate space for a DMA transfer, it must do so by way of a mechanism that guarantees its allocation from DVMA space. There are several ways of making this guarantee:

- `rmalloc()` can be used with the `iopbmap` argument. This will get a small block of memory from the beginning of the DVMA space. Such small blocks of memory are usually used for control information, and not for large



blocks of data.

- For a large buffer, the driver can statically declare a `buf` structure (which is a buffer header that contains a pointer to the data) and then use `mbsetup()` to allocate a buffer for it from DVMA space. This mechanism is primarily intended for block devices but is perfectly adaptable for use by character devices that need large DMA buffers.

When dealing with addresses which are in DVMA space, the driver must strip off the high bits by subtracting the external variable `DVMA`, which contains the address of DVMA (declared as an array of characters). `DVMA` is initialized by the system to either `0x00F00000` (for Sun-2s) or `0xFFF00000` (for Sun-3's, Sun-3x's and Sun-4's). If the driver fails to make this adjustment, the device will attempt to use a null address — in the high megabyte — and the CPU board will not respond to it.

**NOTE** *Addresses received by way of `mbsetup()` (and `MBI_ADDR()`) do not have to be adjusted in this fashion, as `mbsetup()` will have already adjusted them to be relative to the start of DVMA space.*

When the device, in turn, uses the address, the address reference comes down the bus and through a slave decoder, which adds the machine-specific offset to it to map it back into the high megabyte of system virtual memory.

Sun DMA is called DVMA because the addresses which the device uses to communicate with the kernel are virtual addresses like any others. The driver, as part of the kernel, is privy to implementation dependent information, and knows that it must chop off the high-bits of any address intended for the device. This allows the MMU to recognize the addresses destined for the Main Bus and to act accordingly. The device, however, knows nothing of this except that its buffers are mapped to the high megabyte of system virtual memory.

User processes, it should be noted, cannot do DVMA directly into their own address spaces. The kernel, however, provides a way of getting around this limitation by supporting the redundant mapping of physical memory pages into multiple virtual addresses. In this way, a page of user memory (or, for that matter, a page of kernel memory) can be mapped into DVMA space in such a way that transferred data immediately appears in (or immediately comes from) the address space of the process requesting the I/O operation. All that a driver need do to support such direct user-space DVMA is to set up the kernel page maps with the routine `mbsetup()` — the details of the mapping will then be automatically handled by the kernel.

If you wish to do DMA over the Main Bus, you must make the appropriate entries in the kernel memory map. There are two functions, `mbsetup()` and `mbrelse()`, to help with this chore.

## DMA on ATbus Machines

The Sun386i uses the Intel 80386 chip. This chip has an integrated MMU, so the I/O devices cannot access the Sun MMU address-translation facility and therefore must use physical addresses to access memory directly.

To do DMA on the Sun386i, you must make certain changes in the kernel's memory map (its page tables). Use the `mbsetup()`, `dma_setup()`,

`mbrelse()`, and `dma_done()` routines to make these changes. The changes you must make to the kernel memory map are described with these routines in the *Kernel Support Routines* appendix.







---

# Writing Device Drivers





## Driver Development Topics

### 5.1. Installing and Checking the Device

The central processor board (CPU) of the Sun Workstation has a set of PROMs containing a program generally known as the "Monitor". (See the appropriate *PROM Commands* chapter of the *PROM User's Manual* for detailed descriptions of the monitor commands and their syntax). The monitor has three basic purposes:

- 1) To bring the machine up from power on, or from a hard reset (monitor k2 command).
- 2) To provide an interactive tool for examining and setting memory, device registers, page tables and segment tables.
- 3) To boot SunOS, stand-alone programs, or the kernel debugger kadb.

If you simply power up your computer and attempt to use its monitor to examine your device's registers, you will likely fail. This is because, while you may have correctly installed your device (a process that includes specifying its virtual memory mapping in the config file) those mappings are SunOS specific, and don't become active until SunOS is booted. The PROM will, upon power up, map in a set of essential system devices — like the keyboard — but your device is almost certainly not among them.

When installing a new device, you will use the monitor primarily as a means of examining and setting device registers. But before even beginning the development of your driver, it's a good idea to attach your device to the system bus and use the monitor to manually probe and test it. This will give you a chance to become familiar with the details of its operation, and to ensure that it works as you expect it to.

### Setting the Memory Management Unit

Upon power-up, the PROM monitor:

- Maps the beginning of on-board memory, up to 6 megabytes, to low virtual addresses starting at virtual 0x0.
- *Sun-2 machines only.* Maps the bus spaces into virtual address space, for the purpose of supporting Multibus devices. Multibus IO space is mapped from 0xEB0000 to 0xEBFFFF on Sun-2 Multibus machines. On Sun-2 VMEbus machines, vme16d16 is mapped from 0xEB0000 to 0xEBFFFF so that Multibus cards attached by way of VMEbus adapter cards can be accessed. These two address spaces, Multibus I/O and vme16d16, are *not*



remapped by the SunOS kernel. This means, for example, that kernel virtual address 0xEBEE40 can be used to talk to a device at 0xEE40 in Multibus IO space without setting up a mapping. (This shortcut is *only* possible for the two 16-bit Sun-2 spaces).

Later, using the autoconfiguration process, SunOS makes a pass through the config file (actually, through the `ioconf` file that was produced as output by `config` when it processed the config file). For each device, SunOS selects an unused virtual address (using an algorithm that doesn't presently concern us) and maps it into the device's physical address as specified in the config file.

SunOS then calls the `xxprobe()` routine for each device, passing it the chosen virtual address. In this way, `xxprobe()` is kept from having any knowledge of the physical address to which the device is mapped. `xxprobe()` then determines whether or not the device is present. If it isn't, the virtual address can be reused.

To test a device, ignore the SunOS mappings and use the monitor to manually set the MMU to map your device registers to a known address in physical memory. Then you can use the monitor to verify its proper operation. This verification process will consist primarily of using the monitor's O (open a byte), E (open a word) and L (open a long word) commands to examine and modify the device's registers. Note that, in Sun-4 machines, words and long words are both 32 bits in length.

The process of setting up the device for initial testing consists of three discrete steps.

- The selection of an appropriate virtual address for the testing of the device.
- The determination of the physical address of the device, as well as the address space that it occupies.
- The use of the monitor to map the system's virtual address to the device's physical address. Detailed discussion of these three steps follow.

*Since SunOS initializes the MMU in the course of its autoconfiguration process, it's possible to test a device by actually installing it, and then booting and halting SunOS. (You can halt SunOS by pressing the 'LI' and 'A' keys simultaneously, or, on a terminal console, by hitting the <BREAK> key). Having gotten to the monitor by this route, the MMU will be initialized to its SunOS run-time state. You can then use the monitor to test the device, or, if you wish, boot kadb. (A hard reset—the monitor's k2 command—will set the MMU to its pre-SunOS power-up state). But while using the SunOS memory maps may occasionally be useful, it's not what you want to do during the first stages of device integration.*

## Selecting a Virtual Address

First, understand that the MMU, when mapping a virtual address to a physical address, is actually mapping to a page of physical memory and an offset within that page. The low-order bits of a virtual address, those that specify the offset, *do not get mapped*—an address that is X bytes from the beginning of its virtual page will be X bytes from the beginning of whatever physical page it gets

mapped into.

The mapping mechanism is essentially the same for all Sun systems, although the details of address size and page mapping differ. This can be seen in the following diagrams:

Figure 5-1 *Sun-2 Address Mapping*

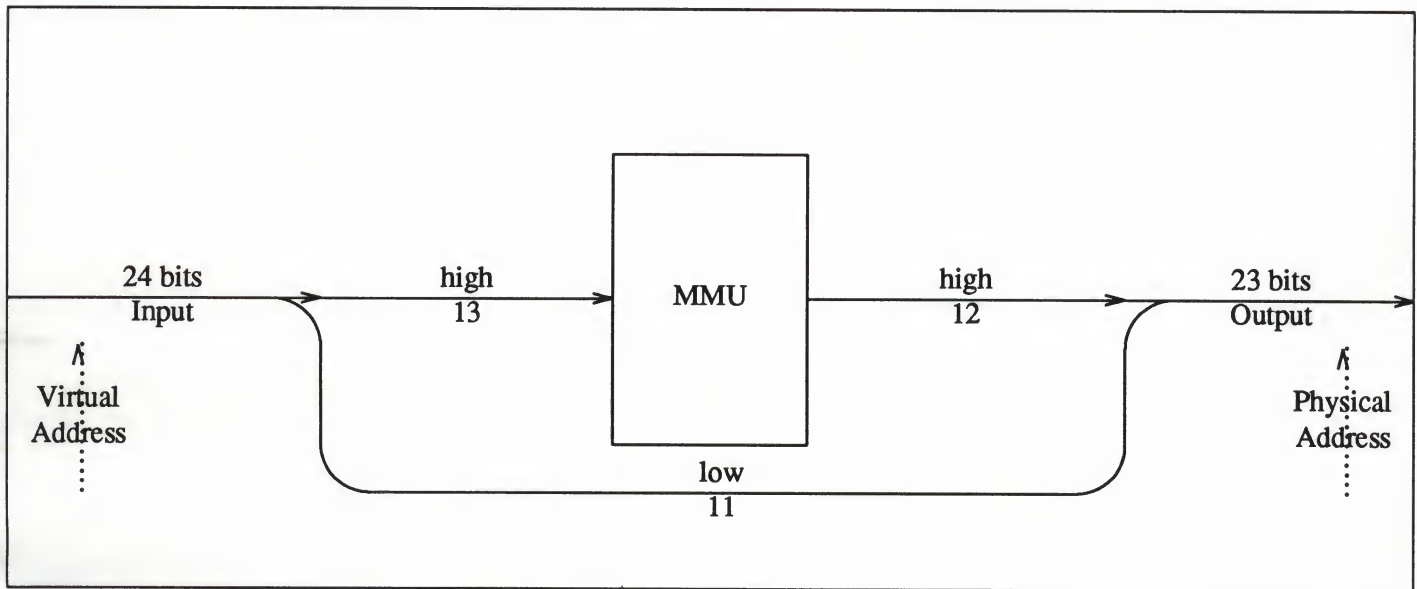


Figure 5-2 *Sun-3 Address Mapping*

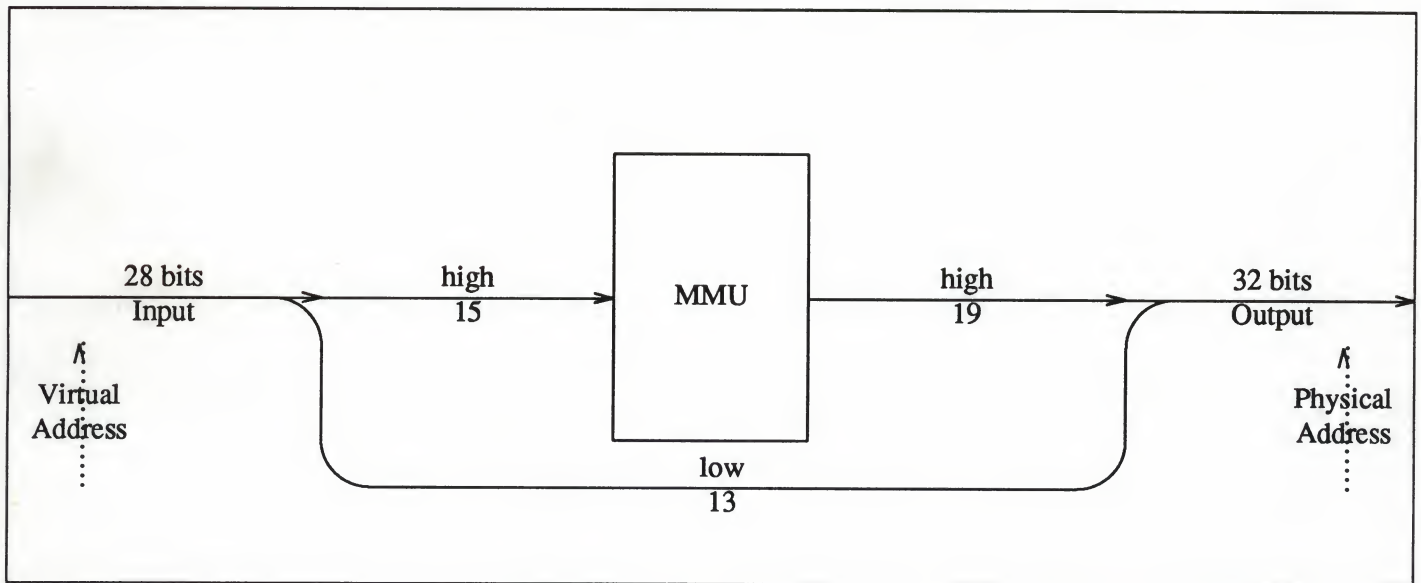
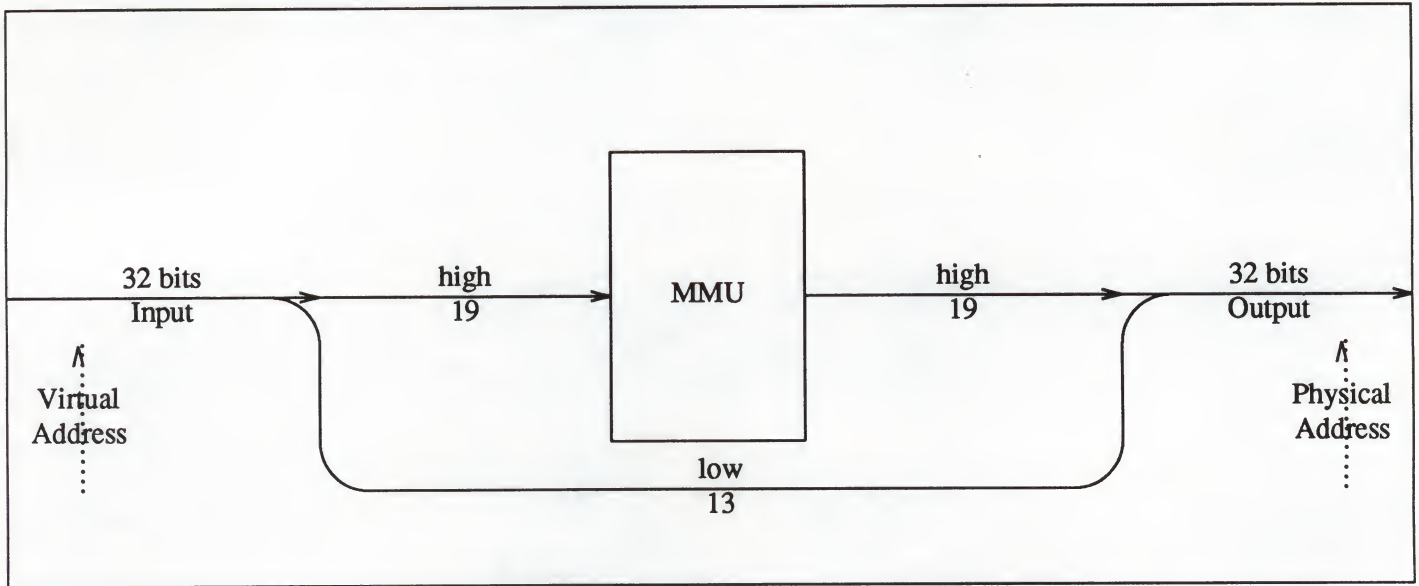
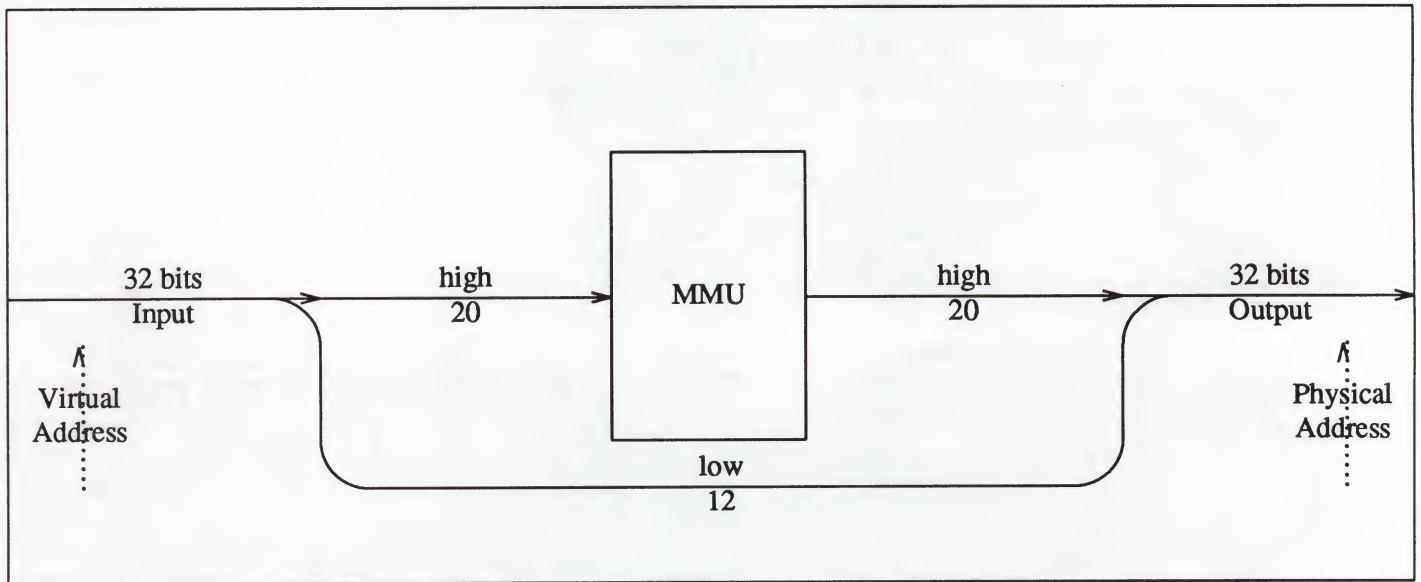




Figure 5-3 *Sun-3x/Sun-4 Address Mapping*Figure 5-4 *Sun386i Address Mapping*

The easiest way to select a virtual address for PROM-monitor testing is to use one between 0x4000 and 0x100000 on Sun-2, Sun-3, Sun-3x and Sun-4 systems, or 0x20000 and 0x100000 on Sun386i systems. Addresses in these ranges are unused by the monitor in the respective Sun models, and are thus

available. (Note that these addresses, while convenient for testing, are not those that the kernel will choose when your device is finally installed).

It's most convenient to select a virtual address which has only zero's in its low-order bits. This way you select the first address in a virtual page. The low-order bits in the address you choose will remain unchanged. With 'X' representing the unmapped low-order bits (11 for a Sun-2, 13 for a Sun-3, Sun-3x or Sun-4, 12 for a Sun386i) the test address 0x4000 is, in binary:

|           |                                         |           |
|-----------|-----------------------------------------|-----------|
| Sun-2 :   | 0000 0000 0010 0XXX XXXX XXXX           | (24 bits) |
| Sun-3 :   | 0000 0000 0000 100X XXXX XXXX XXXX      | (28 bits) |
| Sun-3x:   | 0000 0000 0000 0000 100X XXXX XXXX XXXX | (32 bits) |
| Sun-4 :   | 0000 0000 0000 0000 100X XXXX XXXX XXXX | (32 bits) |
| Sun386i : | 0000 0000 0000 0000 0100 XXXX XXXX XXXX | (32 bits) |

### Finding a Physical Address

Your board may be preconfigured to some address. If it is, then use that address unless it conflicts with the address of an already installed device. If it does, you will have to find an unused physical address at which you can install your device. To do so, examine the kernel config file for the system upon which you are working. Tables in the *Hardware Context* chapter show memory layouts corresponding to typical configurations, but if your system has departed at all from the norm, you will have to consult your kernel's config file (to determine where devices have been installed) and the header files for the corresponding device drivers (to determine how much space they consume on the bus).

### Selecting a Virtual to Physical Mapping

When selecting a virtual to physical mapping, it's best if you understand a bit about the internals of the Memory Management Unit. The Sun-2, Sun-3, and Sun-4 all use the same proprietary MMU architecture. The Sun-3x uses the MMU that is on the same chip as the CPU. This MMU works completely different than the Sun MMU.

The following description is about the Sun MMU operation as it pertains to the Sun-2, Sun-3 and Sun-4. There is also an example of how to perform a mappings using sample numbers. The Sun-3x description follows the Sun-2/ Sun-3/Sun-4 description and includes a page mapping example.

### Sun-2/Sun-3/Sun-4 Virtual to Physical Mapping

Up to this point we've only stressed that the MMU maps the top bits of the virtual address, leaving the offset bits unchanged. Now it will be necessary to explain the mapping process in more detail.

Some new concepts are necessary to discuss the details of virtual to physical memory mapping.

- The *context register* (of real concern only on the Sun-2) is a register specifying which of memory *contexts* should be used when mapping virtual addresses to physical addresses. Sun-2 and Sun-3 Context Registers contain 3 bits, and specify one of eight memory contexts; Sun-4/260 Context Registers contain four bits, and specify one of 16 memory contexts. Each SunOS *process segment* (containing either code, data or stack) is kept within a single memory context.



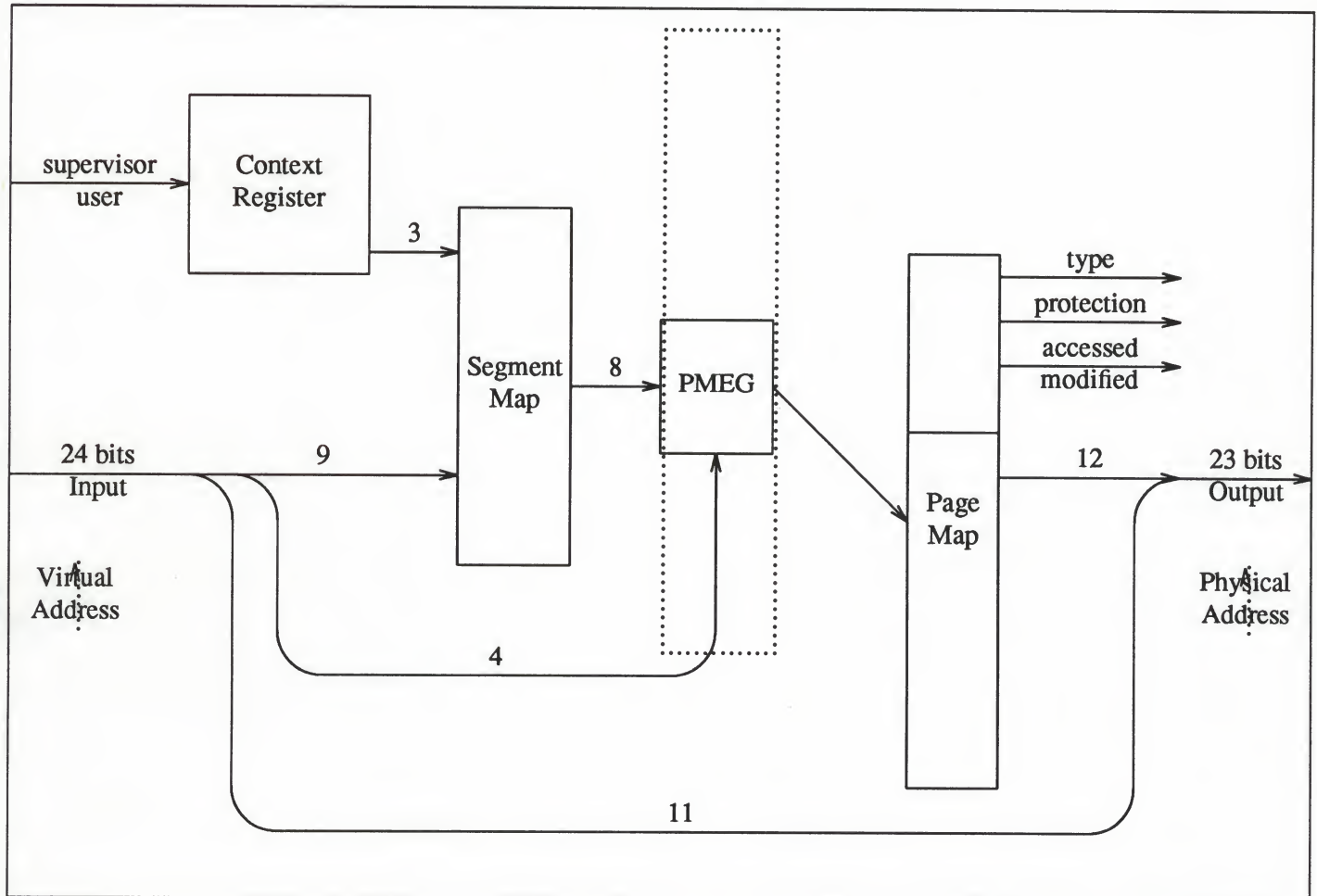
- Sun-3s have user and kernel address spaces in the same hardware context. That is to say, there is only one virtual address space, a portion of which is used by the kernel and the rest by user processes. Sun-4 virtual address spaces are divided into two chunks. One of them is at the top of the addressable virtual memory space and the other is at the bottom. The size of the unused space between these two spaces varies with the model — in the Sun-4/260 each of the two virtual address spaces is 512 megabytes in size, and the space between them consumes 15 Gigabytes.
- Sun-2s, on the other hand, segregate kernel and user processes into separate hardware contexts with separate address maps. Kernel processes are run in the supervisor context (context 0) and only processes in context 0 have access to the I/O devices.
- The *segment map* is used in conjunction with the *context register* to select the *page map entry group* (PMEG) corresponding to the virtual address being mapped. The eight bits in the segment register specify one of a group of 256 PMEGs.
- Within each *page map entry group* there are 16 *page table entries*.
- The *page map* maps the PMEG returned from the segment mapping with a second subfield of the incoming virtual address to exactly specify a single *page table entry* describing the physical page within which the virtual address is mapped.
- The *page table entry* (PTE) is the final output of the MMU. A PTE specifies the physical address of a page, as well as its type (e.g., on-board memory space), protection, and the state of its *access* and *modified* flags.

*Note (for Sun-2 machines only): when testing your device, it's necessary to ensure both that you are in supervisor state and that you are in context zero (the kernel context). The monitor normally initializes to supervisor state, but if you enter it by way of an abort from SunOS, you will remain in whatever context you were in at the time of the abort. To be on the safe side, begin all of your monitor sessions with the command S5. This will put you into supervisor data state, where you want to be. Note one important exception to this rule: if you've `mmap()`'ed the device into your (user) program's address space and want to check that this worked, you must use the S1 command instead of the S5 command. This will cause user function codes to be used when accessing page maps and data.*

## Sun-2 Address Mapping

Note the following diagram of the Sun-2 MMU:

Figure 5-5 Sun-2 MMU



Note that:

- The lower 11 bits of the incoming virtual address are passed through the MMU without being mapped — these are the bits that specify the position within the page, regardless of whether that page is physical or virtual.
- Multiple segment maps can specify the same PMEG, and often do.
- The PTE, on the output side of the MMU, specifies a variety of kinds of status information for the specified page, as well as the top bits of its physical address.

The process of mapping a virtual to a physical address consists, in practice, of plugging the right number into the right PTE. The monitor provides a simple means of addressing the right PTE, but you will have to calculate the right value to plug into it.



On Sun-2 systems, hardware PTEs are 32-bit numbers with the following structure.

|   |   |   |   |   |   |   |      |   |   |            |                      |
|---|---|---|---|---|---|---|------|---|---|------------|----------------------|
| V | r | w | x | r | w | x | Type | a | m | Unused (8) | Physical Page # (12) |
|   |   |   |   |   |   |   |      |   |   |            |                      |

Most of the PTEs that we will deal with will have similar structures, and so we can begin by making a "template" bit mask that we can use to construct our standard PTEs. One acceptable mask will assume values as follows:

```
V (valid) = 1
rwxrwx = 111111
(a/m) accessed/modified = 00
unused = 00000000
```

Thus, we can see that our template will be:

|   |   |   |   |   |   |   |      |   |   |   |   |   |   |   |   |   |   |                      |
|---|---|---|---|---|---|---|------|---|---|---|---|---|---|---|---|---|---|----------------------|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | Type | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Physical Page # (12) |
|   |   |   |   |   |   |   |      |   |   |   |   |   |   |   |   |   |   |                      |

This gives us a mask of 0xFE000000 (if we assume that the type field is 0000). Now, as already mentioned, there are four types of memory, represented in the PTE by values of 0, 1, 2 and 3 in the type field indicated above. (Types 0 and 1 have the same meaning in both Multibus and VMEbus machines, but types 2 and 3 do not.) Type 2 is used, on Sun-2 VMEbus machines, to designate the first 8 megabytes of the 24-bit VMEbus space — 0x0 to 0x7FFFFFFF — and type 3 is used to designate the second 8 megabytes — 0x800000 to 0xFFFFFFF. (But remember that the top 64K of the 24-bit space is stolen for the 16-bit space). This use of two memory types to designate physical memory is necessary because the Sun-2 physical address size, 23 bits, is not sufficient to address all 16 megabytes of vme24d16.

Table 5-1 Sun-2 PTE Masks

| Type | Description             | Mask       |
|------|-------------------------|------------|
| 0    | On Board Memory         | 0xFE000000 |
| 1    | On Board I/O Space      | 0xFE400000 |
| 2    | (Multibus) Memory Space | 0xFE800000 |
| 3    | (Multibus) I/O Space    | 0xFEC00000 |
| 2    | (VMEbus) VMEbus Low     | 0xFE800000 |
| 3    | (VMEbus) VMEbus High    | 0xFEC00000 |

To determine the value which we need to plug into the PTE, we must add the appropriate mask to the appropriate physical page number, thus giving us the full 32-bit number that we need. Here, we will cease to explain details and simply give a series of rules for calculating physical page numbers.

If Sun-2 Multibus:

If Multibus I/O Space, use Type-3 Template  
If Multibus Memory Space, use Type-2 Template

Physical Page Number = Physical Address >> 11

If Sun-2 vme24d16:

If Physical Address >= 0x800000  
Use Type-3 Template  
Physical Page Number =  
(Physical Address - 0x800000) >> 11

If Physical Address < 0x800000  
Use Type-2 Template  
Physical Page Number = Physical Address >> 11

If Sun-2 vme16d16

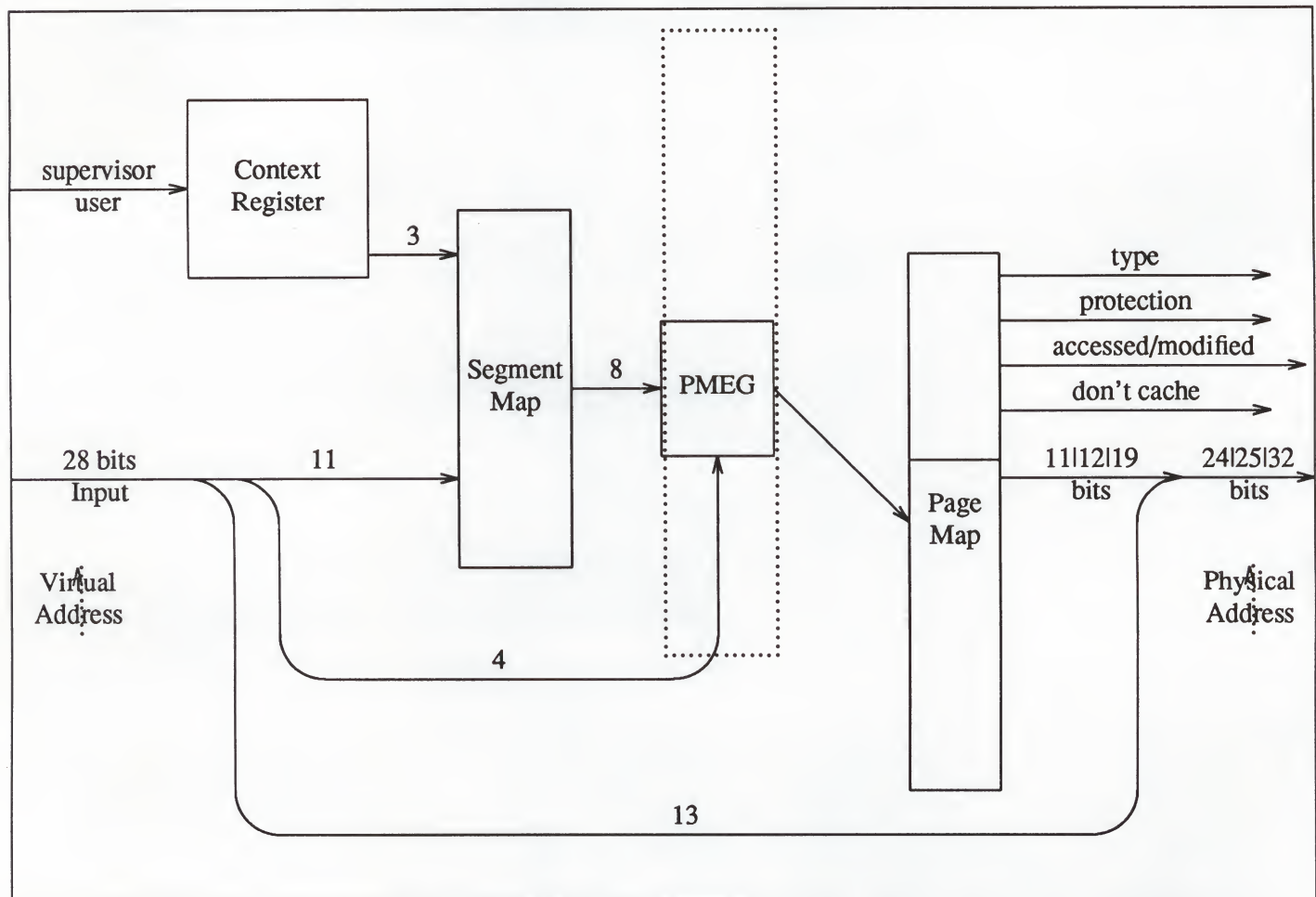
Use Type-3 Template  
Physical Page Number =  
(Physical Address + 0x7F0000) >> 11



# Sun-3 and Sun-4 Address Mapping

Consider the following diagram of address mapping on the Sun-3.

Figure 5-6 Sun-3 MMU

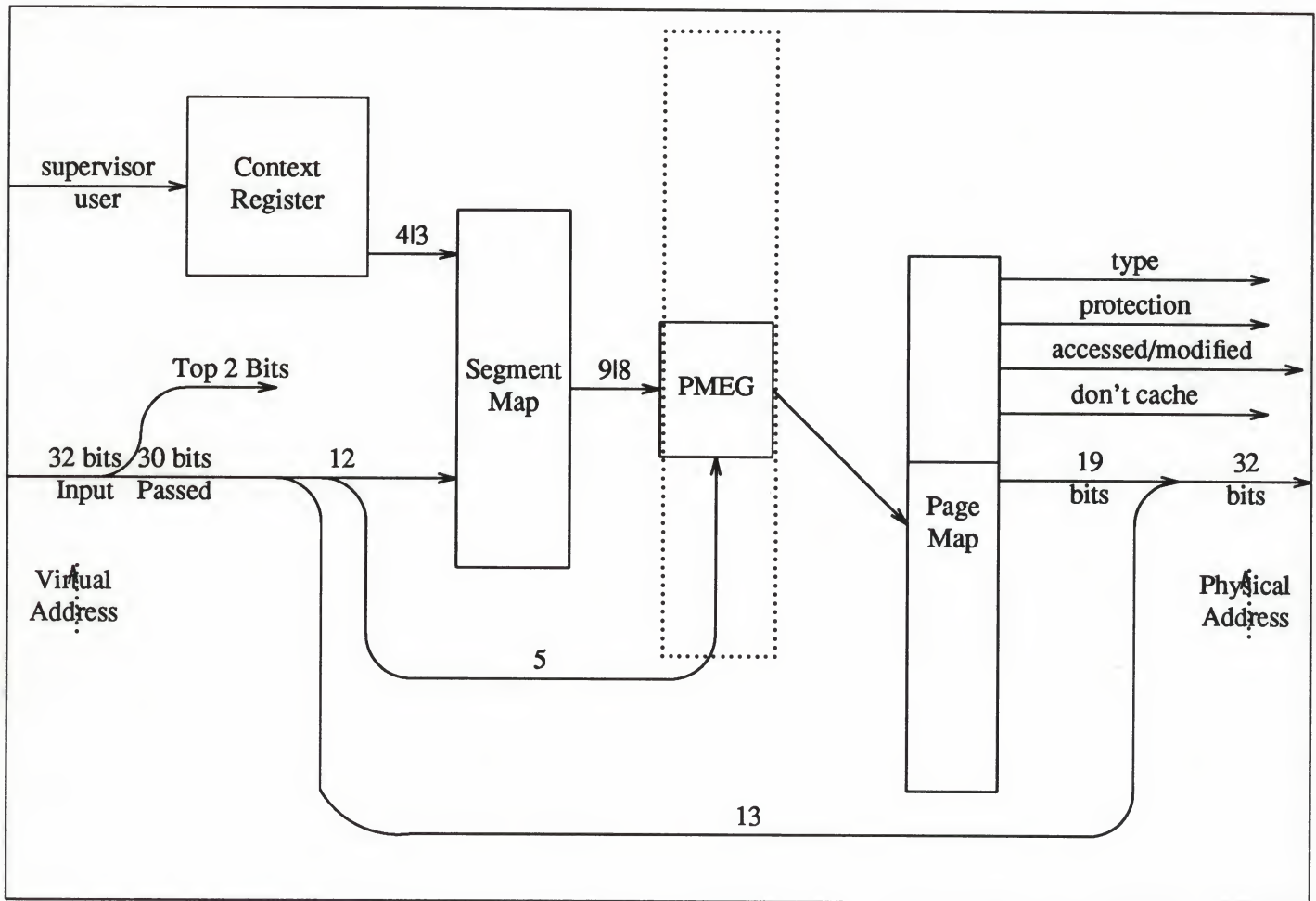


As you can see, the general scheme is the same as it was in the Sun-2, but the details have changed:

- The MMU is getting a 28-bit virtual address as its input, as opposed to a 24-bit address in the Sun-2.
- The number of mode and permission bits in the PTE has been reduced.
- The number of high-order bits reported out of the MMU, and thus the size of the physical address, is variable. The address size is fixed for any given Sun-3 machine, and varies only with the model — there are different kinds of Sun-3 machines and they have different physical address sizes.

The Sun-4 MMU is almost the same:

Figure 5-7 Sun-4 MMU



As you can see, the Sun-4 MMU is largely identical to the Sun-3 MMU. The differences are that:

- The Sun-4 MMU gets a 32-bit virtual address as its input, as opposed to a 28-bit address in the Sun-3. The top two bits are immediately shunted off. They must be either 00 or 11, and are used to specify one of the two “chunks” in the virtual address space. (See *Selecting a Virtual to Physical Mapping* above).
- The number of bits coming off the Context Register is 4 (to specify one of 16 contexts) on Sun-4/260s and 3 (to specify one of 8 contexts) on Sun-4/110s.
- The number of bits coming off the Segment map is 9 for Sun-4/260s and 8 for Sun-4/110s.

On both Sun-3 and Sun-4 systems, PTEs are 32-bit numbers with the following structure.



|   |   |   |   |      |   |   |            |                           |
|---|---|---|---|------|---|---|------------|---------------------------|
| V | w | s | c | Type | a | m | Unused (5) | Physical Page Number (19) |
|   |   |   |   |      |   |   |            |                           |

As we did with Sun-2 PTEs, we will make a “template” bit mask that we can use to construct our standard PTEs. One acceptable mask assumes values as follows:

```

V (valid) = 1
w/s (write ok/supervisor only) = 11
c (don't cache) = 1
(a/m) accessed/modified = 00
unused = 00000

```

(A one (1) in the don't cache position only disables caching if the type is zero (0), since other types of pages are never cached). With the above values, our template will be:

|   |   |   |   |      |   |   |   |   |   |   |   |                           |
|---|---|---|---|------|---|---|---|---|---|---|---|---------------------------|
| 1 | 1 | 1 | 1 | Type | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Physical Page Number (19) |
|   |   |   |   |      |   |   |   |   |   |   |   |                           |

This gives us a mask of 0xF0000000 (if we assume that the type field is 00). Thus, the four masks for the four types of memory are:

Table 5-2 Sun-3/Sun-4 PTE Masks

| Type | Description        | Mask       |
|------|--------------------|------------|
| 0    | On Board Memory    | 0xF0000000 |
| 1    | On Board I/O Space | 0xF4000000 |
| 2    | vme16d16           | 0xF8000000 |
| 2    | vme24d16           | 0xF8000000 |
| 2    | vme32d16           | 0xF8000000 |
| 3    | vme16d32           | 0xFC000000 |
| 3    | vme24d32           | 0xFC000000 |
| 3    | vme32d32           | 0xFC000000 |

To determine the value to be plugged into the PTE, we must add the appropriate mask to the appropriate physical page number, thus giving us the full 32-bit number that we need. Here, again, we will give rules instead of details.

```

If vme16d16
  or vme24d16
  or vme32d16

  Use Type-2 Template

```

```
If vme16d32
    or vme24d32
    or vme32d32

    Use Type-3 Template
```

```
If vme32d16
    or vme32d32

    Physical Page Number = Physical Address >> 13
```

```
If vme24d16
    or vme24d32

    Physical Page Number =
        (Physical Address + 0xFF000000) >> 13
```

```
If vme16d16
    or vme16d32

    Physical Page Number =
        (Physical Address + 0xFFFF0000) >> 13
```

### Sun-3x Virtual to Physical Mapping

In the previous CPU board designs, such as the Sun-3 architecture, a discrete MMU was designed and implemented to handle Demand Paging (off chip). That MMU was implemented mostly in hardware, with a dedicated register for the Context and separate high speed RAM for the Segment and Page values. In the Sun-3x architecture where the MC68030 is used as the CPU, a fully programmable Memory Management Unit (MMU) integrated into the silicon (on the 68030 chip) will be used to handle demand paging. A similar MMU has been offered by Motorola for some time (the MC68851 MMU) but was not used by Sun due to certain architectural incompatibilities.

This Memory Management Unit is drastically different in operation from the popular discrete version of its processors. Some of the MMU's most significant changes involve how the Translation Tables are initialized, accessed, and updated and also the way the Address Translation procedure, or Table Walk, is completed. This next discussion provides the process of how the firmware builds, initializes, and updates the entries in the MMU Translation Tables, how the Table Walk is accomplished, and how the MMU performs Address Translation. An example is shown how to use the monitor to map virtual addresses into physical addresses to access devices through the PROM.

The MMU handles the translation of addresses from virtual to physical using translation tables stored at arbitrary locations in memory. The MMU has an



Address Translation Cache (ATC) that holds recently used virtual to physical address translations. When the CPU passes a virtual address to the MMU for translation, it will first search the ATC for the corresponding physical address. If the requested entry is not in the ATC, the processor will search the translation tables in main memory for the information. An ATC access operates in parallel with the other on-chip caches, namely the CPU's Instruction Cache and Data Cache. In order for the MMU to operate correctly, its internal registers must be initialized to a known state.

The MMU has several internal registers that are initialized to known values before the MMU is Enabled (Address Translation Enabled) and during various Reset (k2 or power-on) operations. These registers include the CPU Root Pointer (CRP), the Supervisor Root Pointer (SRP), and the Translation Control (TC) register, all of which are initialized while the MMU is Disabled (Translation Disabled). The CRP and SRP are discussed in the Motorola 68030 Manual, but for now it is important to say that these registers contain the starting addresses for the MMU's table walk.

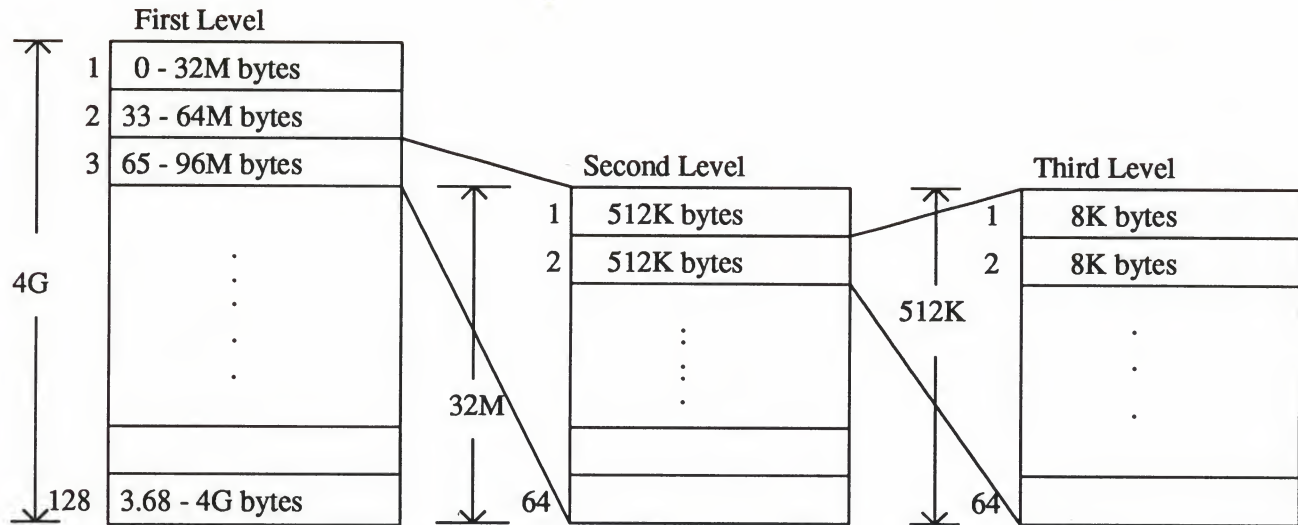
## The Table Walk

The MMU's principal function is address translation, which involves converting a virtual or logical address to a physical address. This process is known as a Table Walk. For the Sun-3x architecture a three level MMU has been designed and requires that a three level table walk be initiated to perform address translation. This process terminates when either an INVALID Entry or PAGE Descriptor is encountered. The three levels of address translation are referred to as TIA, TIB, and PAGE respectively.

The three level table walk is needed to evenly divide the four gigabyte addressing range of the MC68030. This could have been accomplished several different ways, but a specified design goal was to have the Firmware, the Executive Diagnostic and the Unix Operating System all use the same Translation Table format.

The first level of lookup, the TIA table entry, must be able to map in the entire four gigabyte addressing range all at once. The largest block of virtual memory that is required at any one time will be 32 megabytes. By dividing 4 gigabytes by 32 megabytes we get 128 entries for the first level of address translation. For the second level of translation, the TIB entries will take each of the 32 megabyte TIA entries and divide them by 64. This will allow each TIA entry to be accessed as 64 separate 512 Kbytes (1/2 megabyte) blocks. Each of the 64 TIB entries are then divided into 64 again which results in 8 Kbyte page sizes.

It is because of this table traverse that the name Table Walk is used. Each virtual address is translated to a physical one by taking parts of the virtual address and using them as indexes into the three tables, the resulting output being a Page Table Entry (PTE) which will determine that exact physical address. See the table below for how the entire virtual address range is divided into 8 Kbyte ranges.



The beginning of the table walk starts with a pointer to the location of the MMU tables in main memory. The PMMU has two pointers, one that is used by the CPU (CPU Root Pointer), and one that is used by the CPU while in supervisor state (Supervisor Root Pointer). For the firmware's use, both the CRP and the SRP are initialized to the same value, which means they will both point to the base of the MMU tables.

When the MMU is Enabled, the CPU will pass virtual addresses to be translated to the MMU. If the requested entry is not in the ATC, a table walk of the translation table will be initiated. The table walk sequence is described below.

*Step One:* The CRP contains the base address of the TIA table in memory. The top seven bits of the Virtual Address are used to calculate the index into the TIA table. This index is added to the CRP to generate the specific TIA table entry. The TIA entry contains the base address of the TIB table for the next step.

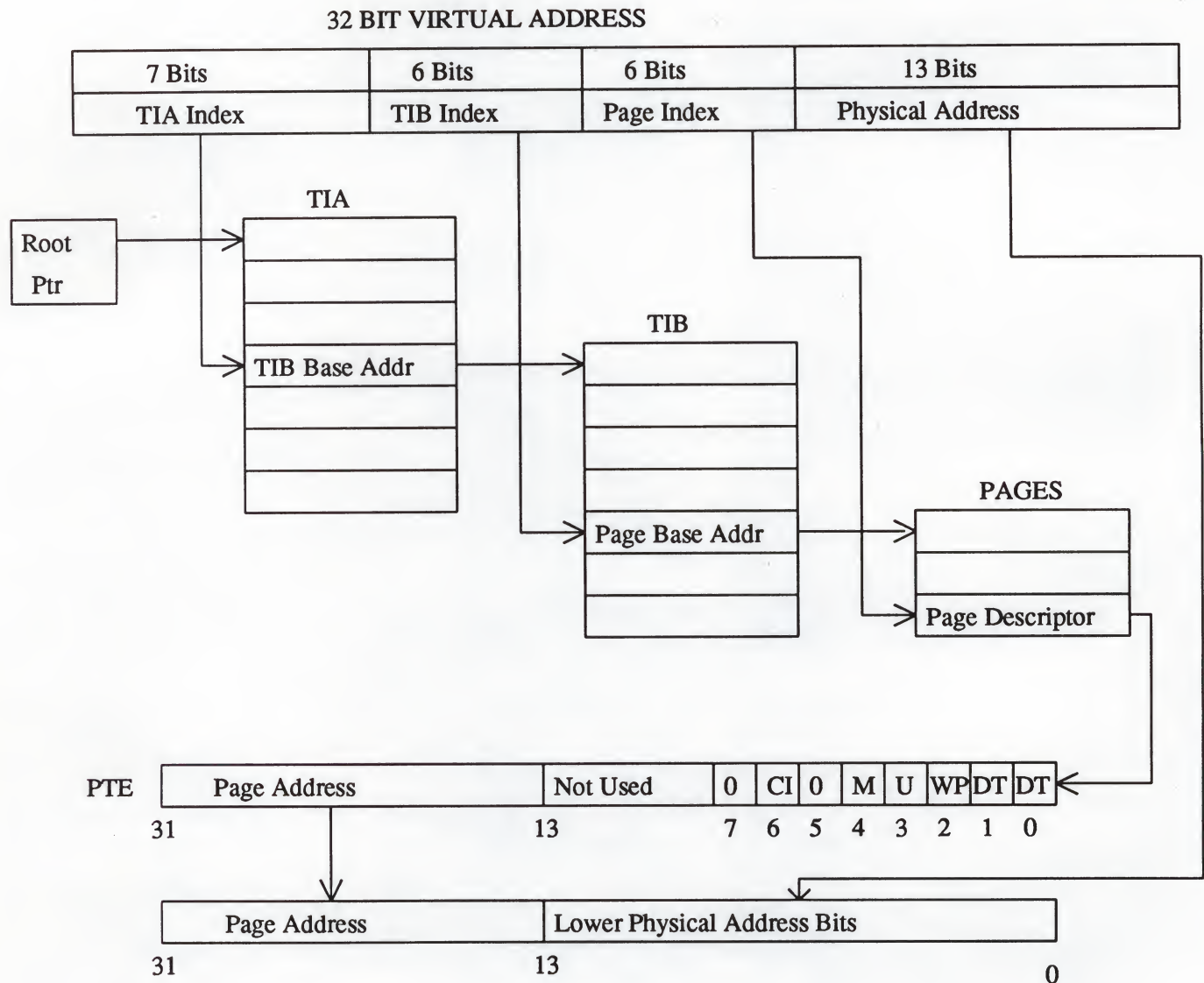
*Step Two:* The next six bits of the virtual address are used as an index into the TIB table. When added to the base address from the TIA table the specific TIB table entry is generated. The TIB entry contains the base address of the PAGE Table.

*Step Three:* The next six bits of the virtual address are used as an index into the PAGE table. The base address from the TIB table plus the index result in the PAGE Table Entry (PTE). The PTE contains a 32 bit PAGE Descriptor of which 19 bits are the Page address, 5 are unused, and the remaining 8 are Status bits.

The Physical address is calculated by taking the top 19 bits from the PTE and the lower 13 bits from the Virtual address. These 13 bits are an offset into the physical memory page that is selected from the 19 bits.

The table walk is completed by passing the physical address back to the CPU. If an INVALID descriptor is ever encountered the table walk terminates.





### A Few Example PTE Calculations

*Example One:* You wish to map a device which you have attached at physical 0x280008 onto bus type vme24d16 which will be mapped into virtual memory at address 0xE000000. What is the corresponding PTE?

### Sun-3 Solution

Well, since we are mapping the device into vme24d16, we will use 0xF8000000 as the template. Then, following the Sun-3 rules, as given above, we add the physical address to 0xFF000000. This yields 0xFF280008. In binary, this is:

1111 1111 0010 1000 0000 0000 0000 1000

Shifting this right by 13 yields:

XXXX XXXX XXXX X111 1111 1001 0100 0000

Adding the template, 0xF8000000, we get values for the 13 bits that are undefined from the shift. Thus the PTE is:

```
1111 1000 0000 0111 1111 1001 0100 0000
```

Which is 0xF807F940.

A final note: we've now calculated the PTE that maps the virtual page beginning at 0xE000000 to the physical page containing 0x280008. To get the virtual address by which to access the device it's necessary to take the lower 13 bits of the physical installation address — the bits that are just passed through the MMU — and add them to virtual 0xE000000. The lower 13 bits of physical 0x280008 are 0008, and adding them to 0xE000000 yields 0xE000008, the virtual address by which the device can be accessed.

#### Sun-3x Solution

Our variables are:

|                  |          |
|------------------|----------|
| physical address | 280008   |
| virtual address  | E000000  |
| bus type         | vme24d16 |

The base address for vme24d16 for the Sun-3x, which is in Table 2-8 in Chapter 2, is 0x7e000000. So we add the physical memory address to the vme base pointer which gives us a specific physical address.

|          |          |
|----------|----------|
| vme24d16 | 7E000000 |
| physical | 280008   |
|          | -----    |
| physical | 7E280008 |

Then we take off the top 19 bits to mask out just the vme page, which gives us the physical page of memory. We then need to logically 'or' in some status bits to allow us to write to this page. The value 1 enables the write status.

|          |          |
|----------|----------|
| physical | 7E280008 |
| and mask | 7E280000 |
|          | -----    |
| page     | 7E280000 |
| or flag  | 1        |
|          | -----    |
| PTE      | 7E280001 |

To use the monitor to perform the mapping, use the 'p' command for displaying and changing the Page Table. The syntax is

```
p[virtual address]
```

where the virtual address is the original virtual memory given in the problem initially. The monitor returns the current PTE and asks you for a new value. The newly calculated PTE is input, which modifies the PTE to map to a new physical memory location



```
monitor cmd      >pE000000<cr>
return value     xxxxxxxx
new PTE          ?7E280001<cr>
exit monitor     .
```

Now every reference to the virtual memory location E000008 will be mapped to the device. Note that since the original physical address was folded into the virtual address and then was masked, we still have the 8 offset at the end of the memory reference to index into the physical page of memory to access the device.

*Example Two:* You wish to map physical 0xEE48 on bus type vme16d32 on a Sun-3. Using virtual address 0xE000000, what is the PTE?

#### Sun-3 Solution

Since we are mapping the device into vme16d32, we will use 0xFC000000 as the template. Then, following the Sun-3 rules, as given above, we add the physical address to 0xFFFF0000. This yields 0xFFFFEE48. In binary, this is:

```
1111 1111 1111 1111 1110 1110 0100 1000
```

Shifting this right by 13 yields:

```
xxxx xxxx xxxx x111 1111 1111 1111 1111
```

Adding the template, 0xFC000000, we get values for the 13 bits that are undefined from the shift. Thus the PTE is:

```
1111 1100 0000 0111 1111 1111 1111 1111
```

Which is 0xFC07FFFF.

To get the virtual address by which to access the device at physical 0xEE48, add its lower 13 bits, 0xE48, to 0xE000000 — this yields 0xE000E48.

#### Sun-3x Solution

Using the same steps above, this is how the solution looks:

```

physical      EE48
virtual       E000000
bus type      vme16d32

vme16d32      7D000000
physical      EE48
-----
physical      7D00EE48

physical      7D00EE48
and mask      7D00E000      0111 1101 0000 0000 111
-----
masked page   7D00E000
or flag       1
-----
PTE           7D00E001

```

This is the new PTE value that can be used in the monitor as shown in the previous example.

### Getting the Device Working and in a Known State

Before you even *think* about writing any code you should check out your device. You must get to know it, finding out early if it has any peculiarities that will affect its driver. It may, for example, have addressing and data-bandwidth limitations. Or, if it's a bus master, it may not implement the *release on request* bus-arbitration scheme the Sun supports. *Know the peculiarities of your device early, and then test it to verify that it's working before proceeding further with driver development.*

Make sure that the board is set up as specified in the vendor's manual. Device characteristics which, in general, have to be set properly before the device can successfully be used include:

- I/O register addresses for I/O mapped Multibus boards,
- Memory base addresses for Multibus boards that use Multibus memory space,
- Address and data widths,
- Interrupt levels,
- Interrupt vector numbers for VMEbus device,
- VMEbus address modifiers,
- The bus grant level for VMEbus devices should be set at 3.

Then, take down your system and power it off. Plug the device into the card cage and attempt to bring the system back up. If you can't boot the system, then there's a problem. Perhaps the board isn't really working, or perhaps it's responding to addresses used by other system devices. You must resolve this problem before proceeding further.



Take SunOS down again and attempt to contact the device using the PROM monitor. To do so, you will need to set up a PTE on the Sun-2, Sun-3, or Sun-4 which maps to the device's physical installation address. Use the procedures given above to calculate a PTE, then:

- Issue the monitor command that puts you into *supervisor data* state. This will be **s B** for Sun-4 machines and **s5** for all others. So, if you have a Sun-3, give the

**>s5**

command.

- Calculate, using the procedures given above, the PTE appropriate to the physical address you've chosen.
- Set the position in the kernel page map that corresponds to your physical address to contain the calculated PTE. This will map your chosen physical address, thus putting you in contact with your device. You may use the monitor's **p** command to perform this mapping. The **p** command takes a virtual address as its argument, displays the PTE that corresponds to that virtual address, and gives you the option of modifying the PTE. For example:

**>pF32000**

selects the page map entry that corresponds to the virtual address of 0xF32000 and displays it. It also displays a '?', which indicates that you may type in a new value to replace the one displayed. (See the appropriate *PROM Commands* chapter of the *PROM User's Manual* for more details). Note that all virtual addresses within a page select the same PTE.

Having contacted the device from the monitor, try some of the following:

- Try reading from the device status register(s), if there are any.
- Try writing to the device control and data registers(s), if there are any. Then try reading the data back to see if it got written properly (this assumes, of course, that the device allows the reading of these register(s)).
- Try actually getting the device to do something by sending it data.
- If the device is a controller with separate slave devices, then switch a slave on and off and watch for changes in the controller status bits.

Your goal is to try to actually operate the device, for a moment, from the monitor. For example, if you have a line printer, try to print a line with a few characters. Be aware that bit and byte ordering issues are critical in this process. The reason you're doing this is to ensure that the device works and that you understand the way it works. When you understand the device's peculiarities, you can proceed to write a driver for it.

## A Warning about Monitor Usage

When you use the monitor's `o`, `e` or `l` commands to open a location, the monitor *reads* the present contents of that location and displays them before giving you the option to rewrite them. In the best of all possible worlds, this would present no problems, *but many devices don't respond to reads and writes in as straightforward a fashion as does normal memory.*

For example, the Intel 8251A and the Signetics 2651 use the same externally addressable register to access *two* separate internal mode registers, and they have internal state logic that alternates accesses to the external register between the two internal registers. So suppose that you want to put something in mode register 1 of the 8251. You open the external register, the monitor displays its contents, and you then do your write. If, being cautious, you then read the external register to check that the data you wrote is there, you will find that it's not — because the read will sequence you on to the second register.

To deal correctly with such devices, it's necessary to use the monitor's "write without looking" facility and then read the locations back later to check them. You can write without looking with any of the monitor commands that "open" an area of memory; all that's necessary is that you enter a value after the address argument. For example:

```
>l [address] [value]
```

This will cause `value` to be written into `address` without first reading its current contents. For more information on hardware peculiarities and the problems that they can cause for the monitor, the *Hardware Peculiarities to Watch Out For* section of the *Hardware Context* chapter.

## 5.2. Installation Options for Memory-Mapped Devices

### Memory-Mapped Device Drivers

Memory-mapped devices are the simplest types of devices to write drivers for. Frequently, however, their essential simplicity isn't obvious from a quick glance at their source code. This is because many memory-mapped devices are frame buffers, and frame-buffer drivers must set up and manage the low-level interface for the Sun window system as well as the standard device interface. Consequently, they tend to be littered with declarations and manipulations related to the "pixrect" (pixel rectangle) system. See the *Pixrect Reference Manual* for more details.

Memory-mapped devices are most frequently installed into Sun systems with simple drivers that map them into user address space (there are sometimes alternatives to such drivers, as you will see below). Such memory-mapped drivers don't really do much. Obviously, `xxprobe()` and `xxmmap()` must exist, for the kernel must be able to check the device installation and perform the actual device mapping. And, in addition, `xxintr()` must be real if the device is interrupt driven. But `xxopen()` and `xxclose()` are usually stubs, and `xxread()` and `xxwrite()` can be calls to `nulldev`.



Keep in mind that the major purpose of a memory-mapped driver is to support the `mmap()` system call. This is very important because user processes which call window code must first map the frame buffer into their address space. They do so with the `mmap()` system call, which is translated by the kernel into a series of calls to the driver's `mmap` routine. Each of these calls returns page table entry information which the kernel needs to map a single page (the next page) of frame-buffer memory into a virtual address space. Here's some very simple driver `xmmap()` code.

```

/*ARGSUSED*/
cgonemmap(dev, off, prot)
    dev_t dev;
    off_t off;
    int prot;
{
    return (fbmmap(dev, off, prot, NCGONE, cgoneinfo, CG1SIZE));
}

/*ARGSUSED*/
int fbmmap(dev, off, prot, numdevs, mb_devs, size)
    dev_t dev;
    off_t off;
    int prot, numdevs;
    struct mb_device **mb_devs;
    int size;
{
    int kpfnum;

    if ((u_int) off >= size)
        return -1;

    kpfnum =
        hat_getkpfnum(mb_devs[minor(dev)]->md_addr + off);
    return kpfnum;
}

```

*dev* is, of course, the device major and minor number, and *off* is the offset into the frame buffer (passed down from the user's `mmap()` system call). *prot* is also passed down from the user's call, but it is not currently used. As you can see, there's a bit of shuffling around and then a call to `hat_getkpfnum`, which returns a Page Frame Number which `xmmap()` is expected to return.

Note that `mb_dev->md_addr` is the address of the frame buffer from the Main Bus device structure. This is the device installation address as given in the kernel config file. The offset is checked to be sure the user isn't mapping beyond the end of the frame buffer.

## Mapping Devices Without Device Drivers

Under a restricted set of circumstances, it's possible to avoid writing a device driver altogether by using the `mmap()` system call to overlay the device's registers and memory onto user memory. Having done this, you can read and write the registers — as if they were normal user memory — from a user program.

What this really amounts to is piggybacking the new device onto an another, system standard, virtual memory device (and its driver). The `mmap()` routine of a system virtual memory device is then used to do the user-device mapping, and the "installation" is accomplished without the development of a driver specific to the user device. Instead, a user level program is written, one that calls the `mmap()` system call.

The restrictions on this shortcut are, however, fairly severe.

- The device must not require any special handling of the type that would go into `xxioctl()`.
- The device (including all its control registers) must work with user function codes, since that's what it will get when mapped into and then accessed from user space.

**NOTE** *MC680X0 processors, SPARC processors and the Intel 80386 all run in either 'user' or 'supervisor' state. Many devices, in turn, restrict certain of their operations, and will only perform them when the processor is in supervisor state. The Sun CPU is in supervisor state only when executing kernel code. This means that device drivers, which are part of the kernel, can issue device commands which are not available from user processes. Also note that, when the CPU is in supervisor state, as it is when driver code is executing, the device will receive different VMEbus address modifier codes than when the CPU is in user state. For details about these codes see the VMEbus specification.*

- The device must not require any other sort of special handling — it cannot, for example, be multiplexed, interrupt driven, or do DMA.
- Finally, there are security problems associated with this sort of installation. Since the system virtual-memory devices are normally owned by and restricted to the superuser, your programs will either have to change their permissions to allow normal users to access them, or will have to run with superuser privileges. The former strategy is usually not acceptable in the long run, because it creates a gaping hole in the security of the system. And it's far from clear that the second alternative is desirable either.

The virtual-memory devices of interest here are those that support mapping over the entire range of a virtual address space. They are:



Table 5-3 Virtual Memory Devices

| Machine Type                | Memory Device Name |
|-----------------------------|--------------------|
| Multibus (Sun-2 only)       | mbmem              |
| Multibus (Sun-2 only)       | mbio               |
| VMEbus (All Sun's)          | vme16d16           |
| VMEbus (All Sun's)          | vme24d16           |
| VMEbus (Sun-3 and Sun-4)    | vme32d16           |
| VMEbus (Sun-3/Sun-3x/Sun-4) | vme16d32           |
| VMEbus (Sun-3/Sun-3x/Sun-4) | vme24d32           |
| VMEbus (Sun-3/Sun-3x/Sun-4) | vme32d32           |
| ATbus (Sun386i only)        | atmem              |

In addition, there are memory pseudo-devices that support access to the on-board devices that users are allowed to access. These are `/dev/fb`, `/dev/mem` and `/dev/kmem` (See the `mem(4)` manual page for details).

`/dev/fb` is a memory device which, on any given system, is set up to address the local frame-buffer device. It can be used as if it were a system memory device — on any given system, `/dev/fb` can be `mmap()`'ed into user memory and then written to, with the effect of writing the local frame buffer memory.

To use `mmap()` with one of the system memory devices, you must do three things:

- Open the device.
- Calculate the *offset* which you will need to call `mmap()`. This offset is merely the device address on the appropriate system memory device rounded to a page boundary. That is to say that you get the offset from the device manual and/or the switches on the device itself.
- Call `mmap()` to allocate virtual space and map in the physical bus address of your device, which you must know. (See the *Hardware Context* chapter for a discussion on how to pick a good physical address from the information in the system config file).

The following example program uses `/dev/fb` rather than one of the virtual memory devices. It makes a good example because it maps the system frame buffer into user memory so that it can then be written from a user program. It uses `mmap()` to set things up, but doesn't bother with calling `munmap()`, because unmapping occurs automatically when the memory device is closed. This close occurs implicitly when the program ceases execution. (The machine segment size is 128K for the Sun-2 and Sun-3; 256K for the Sun-4; and 4Mbytes for the Sun386i. Areas greater than the machine segment size should be mapped only with special care. The Sun-3x has no segment size so any input value will work. For details, see the discussion of `mmap()` in the *User Support Routines* appendix).

Once the device has been mapped into user space it can be treated as a piece of local user memory. (Remember that memory accesses performed by way of this mechanism will be reflected — at the device level — as non-privileged (user) accesses. This is because `mmap()` accesses inherit the privilege of the process that calls `mmap()`. Thus, if memory is mapped by a driver, subsequent accesses to it will have the standard supervisor data access privilege, but if it's called from a user process, as described here, subsequent accesses will be non-privileged. Attempts to access supervisor-only device registers without supervisor privilege might produce a bus error, i.e., they're inaccessible from a user program, and thus a kernel level driver must be written to manipulate them. The device will also receive different address modifier codes when accessed from a user process than when accessed via a device driver).

```
#include <stdio.h>
#include <sys/file.h>
#include <sys/mman.h>
#include <sys/types.h>

/* Width and Height of Frame Buffer in Bits */
#define WIDTH 1152
#define HEIGHT 900

main()
{
    int fd;
    unsigned len;
    char *addr;

    /* Open the frame-buffer device */
    if ((fd = open("/dev/fb", O_RDWR)) < 0)
        syserr("open");

    /* Compute total number of bytes */
    len = ((WIDTH * HEIGHT) / 8);

    /*
     * offset must be page aligned. /dev/fb
     * is already aligned with frame-buffer memory
     */
    offset = 0;

    /* Map device memory to user space */
    addr = mmap((caddr_t)0, len, PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
    if (addr == (caddr_t)-1)
        syserr("mmap failed");

    writeFB(addr);
    exit(0);
}
```



```

writeFB(addr) /* Write to frame buffer */
char *addr;
{
    char color;
    int i,j;

    color = 0xFF;
    for (i = 0; i < HEIGHT; i++) {
        color = ~color;
        for (j = 0; j < WIDTH/8; j++)
            *addr++ = color;
    }
}

syserr(msg) /* print system call error message and terminate */
char *msg;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[];

    fprintf(stderr, "ERROR: %s (%d", msg, errno);
    if (errno > 0 && errno < sys_nerr)
        fprintf(stderr, "; %s\n", sys_errlist[errno]);
    else
        fprintf(stderr, ")\n");
    exit(1);
}

```

**NOTE** *This example uses the special memory device /dev/fb, since this device is always set up to address the frame buffer memory.*

So, despite the plethora of limitations on the sorts of devices that can be installed by way of mapping them into user space, it's quite an easy thing to do. If your device characteristics are such that this is an option, you may well wish to take it. And even if such an installation isn't an attractive long-term option (for example, because of unacceptable security problems) it may still be attractive as a short-term alternative to driver development. Even in environments where security considerations make it unacceptable in the long term, it can allow you to get your device up and running very quickly. Sometimes this counts for a lot.

## Direct Opening of Memory Devices

It should be noted, for the purpose of completeness, that there's another approach to avoiding driver development, one that's even easier than the use of `mmap()` described here, and even more limited. That is, it's possible to simply open the virtual memory device that contains your board, to seek to the location of its registers, and then to read and write those registers as if they were regular memory.

*This approach has most of the same problems as does the use of `mmap()`, and is notable mainly because, with it, the device receives supervisor function codes. It does, however, introduce new problems. It doesn't give you the same degree of control as does `mmap()`, and you often need that control when dealing with*

devices. When you use `mmap()`, the device actually becomes part of your user memory space, and it's left to the compiler to generate exactly the I/O accesses which you implicitly specify in your structure and variable declarations. You can always access exactly what you want, and the accesses occur directly as *move byte* and *move word* operations. Thus they are very fast.

When, however, you simply open a system memory device as a file and then read and write to it, your communication with your board is mediated by the I/O system. The I/O systems will always try to do the "right thing" (if you request I/O at an odd address or for an odd number of bytes it will perform byte access as appropriate; otherwise it will use short integers), but it still doesn't give you the kind of control that can be had using `mmap()`. Furthermore, I/O operations involve lots of code, and take *hundreds* of times as long as direct references to `mmap()`ed references, which proceed by way of the MMU and use low-level store and move instructions to directly access device registers and memory as physical memory.

So the bottom line is that, unless you need to access a device only a few times, or if you need to receive supervisor function codes (and the corresponding VMEbus address-modifier codes) and performance isn't critical, you can do your installation by opening a system memory device and then seeking to your device registers and memory space. Otherwise, use `mmap()` or write a driver. If you do decide to use the `open()/lseek()` method, do so with low-level I/O rather than with the standard I/O library. The standard I/O library implements a buffered I/O scheme which will add considerably to your problems.

The following user program is similar to the example above, in that it writes the same pattern to the memory of a frame buffer. This time, though, the write is done by way of the I/O system rather than by using `mmap()`, and the frame buffer is taken to be installed at *OFFSET* (whatever the device physical installation address is) in the `vme24d16` memory space.

**NOTE** *Since all Sun VMEbus machines have a built-in, on-board frame buffer, this example is only meaningful for color frame buffers. On Sun-2 Multibus machines, however, this code would work with `/dev/obmem` and an offset of `BW2MB_FB`.*

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/buf.h>
#include <sys/file.h>

void syserr();
long lseek();

/* Width and Height of Frame Buffer in Bits */
#define WIDTH 1152
#define HEIGHT 900

main()
{
```





```

int fd;

/* Open the system memory device containing the frame buffer */
if ((fd = open("/dev/vme24", O_RDWR)) < 0)
    syserr("open");

/* Seek to the frame buffer memory */
if (lseek(fd, (long)OFFSET, L_SET) == -1L)
    syserr("lseek");

writeFB(addr);
exit(0);
}

writeFB(fd) /* Write to frame buffer */
int fd;
{
    char color;
    int i, j;

    color = 0xFF;
    for (i = 0; i < HEIGHT; i++) {
        color = ~color;
        for (j = 0; j < WIDTH/8; j++) {
            if (write(fd, &color, 1) == -1)
                syserr("write");
        }
    }
}

```

### 5.3. Debugging Techniques

As described above, it's a good idea to begin debugging by using the monitor to check that the device has been installed at the intended address, and that it works, before proceeding to debug your device driver. This allows you to avoid debugging the device simultaneously with the driver, and experience that you'd like to avoid for as long as possible. Alternatively, if you're confident in both your device and the correctness of your installation, you can simply make a new kernel, boot it and proceed with debugging. In this case you should put some `printf()` messages — see below — into the `xprobe()` routine. Then you can at least see the device get contacted and initialized.

Debugging drivers is significantly more difficult than debugging regular user programs, for a number of reasons:

- In the first place, device drivers are part of the system kernel. This means that the system is not protected from their errors. Addressing errors, for example, will frequently trip hardware traps and crash the system.
- As mentioned above, there's the possibility that the device hardware will be buggy. For this reason, you can't really trust your environment in the same way as you can when writing a user program on a mature computer system.

- Some devices behave in rather peculiar ways. (See *A Warning about Monitor Usage*, above).
- Finally, the debugging environment in the kernel is thinner than it is in user space. There is a kernel debugger, `kadb`, and this a big step towards making life easier for driver developers. Still, life remains more difficult when debugging in kernel space.

*It's possible to prototype drivers in user address space by using techniques similar to those described in the Mapping Devices Without Device Drivers section of this chapter. The same constraints given there apply to prototyping. In particular, it's not possible to run an interrupt routine, or to probe for non-existent devices without generating bus errors from prototype drivers in user space. If the device generates no interrupts, and if it doesn't do DMA, the entire driver might be able to run in user space.*

For all these reasons, you should give extra care to desk-checking your code, and check a reference manual when not absolutely sure of the meaning of a given construction. Don't take chances.

Also, make changes incrementally. Don't try to save time by making many changes at once. You will save time in the long run if you take the time to add and test a few parts at a time. Keep your feet on solid ground.

Use trace output from `printf()`, as described below. Drivers can act in surprising ways, and the best way to proceed is by making the flow of operations highly visible.

**NOTE** *On the Sun386i system, the loadable drivers feature makes driver development much easier because the code-compile-reboot-test cycle is reduced to code-compile-load-test.*

### Debugging with `printf()`

With the availability of `kadb`, the kernel debugger, the importance of `printf()` in the debugging of device drivers has been significantly reduced. Still, even with `kadb` available, `printf()` statements remain useful as means of providing synchronous tracing of overall driver flow and structure. `kadb` can be made to provide a similar sort of tracing (by tying print commands to strategically chosen breakpoints) but this won't altogether eliminate the `printf()` statement. The `printf()` has long found application in driver debugging, and, as a matter of taste and experience, some programmers will continue to use it. For this reason, we will discuss its use in some detail.

The kernel `printf()` sends its message directly to the systems console, without going through the tty driver. As a consequence, the printing is uninterruptible—the characters aren't buffered. Furthermore, `printf()` runs at high priority, and no other kernel or user process activity takes place while its output is being produced. `printf()` thus radically limits overall system performance (though this is usually ok while device drivers are being debugged).

The window systems should not be up when you use `printf()` to debug a driver because its output will go to the console window. On



the Sun386i system, it is best to set the global variable `newlog` to 0.

There is a second kernel print statement, `uprntf()`. `uprntf()`, however, is of little use to driver developers. It attempts to print to the current user tty as identified in the user structure, and prints to the console only if there's no current user tty (at which point it becomes identical to `printf()`). `uprntf()` cannot be called from lower-half routines, which run in interrupt context and cannot make any assumptions about the user structure (where `uprntf()` looks to determine the current user tty). `uprntf()` is most useful for production drivers, like tape drivers that encounter media errors, which want to report errors not to a programmer but to the user.

*There are occasions in which the use of `printf()` (or `uprntf()`) statements will change the behavior of your driver. `printf()` statements, for example, can affect the timing of operations in the driver being tested as well as in other drivers. The output may be so slow relative to other device operations that interrupts are lost and system failures are introduced; thus, it is frequently impossible to synchronously trace a device interrupt routine. Driver code may begin to fail only when `printf()`s are introduced, or, even worse, only when `printf()`s are disabled. If you're debugging a tty driver, you may even face a situation where `printf()`-based tracing generates new calls to the driver being debugged. Thus, there are situations in which it cannot be used. In such situations, you should use `kadb` or the techniques suggested below in the section on Asynchronous Tracing.*

The best way to use `printf()` statements for tracing driver execution is by setting things up so that you can toggle printing by using the kernel debugger, `kadb` (see below) to set and reset print-control variables. Doing so is very simple. At the top of the driver source file, include statements like:

```
#ifdef XXDEBUG
int xxdebug = 0;
#define XXDPRINT if (xxdebug > 0) printf
#endif
```

(It's important that the variables like `xxdebug` be global, so that you can later access them freely from the debugger — remember that all drivers are part of one program, the kernel, and name your print-control variables so as to avoid naming conflicts).

Then, instead of calling `printf()` inside the driver routines, call `XXDPRINT`. Each call should be in the form:

```
#ifdef XXDEBUG
XXDPRINT("driver name...", ...);
#endif
```

which will only call `printf()` if `XXDEBUG` is defined and `xxdebug` is set to a value greater than 0.

Make sure that each call to `XXDPRINT` identifies the driver, for it's possible that you, or some other programmer, will want to see debugging output from several drivers at once. And leave the debugging code in for a while after you're

finished — bugs may surface later.

Having set things up like this, you can turn the `printf()` 's on or off at any time by using `kadb` to set unset or change the print-control variable `xxdebug`. Or you can use `adb` if you wish, running it at user level in a separate window:

```
example adb -w /vmunix /dev/kmem
```

(`adb` won't allow you to set breakpoints in the kernel, but it will allow you to set and unset variables — you can change the value of `xxdebug`, or even reset a variable which has caused your driver to hang). *Remember that you're in the kernel and BE CAREFUL.*

Incidentally, `/dev/kmem` represents the kernel *virtual* address space, which is why it's used here. `adb -k /vmunix /dev/mem`, in contrast, generates a view of the *physical* address space, because `/dev/mem` represents the physical memory. This latter command is useful for examining core files.

Good places to put `printf()` statements include:

- driver routine entry points
- before critical subroutine calls
- upon reading status information from the device
- before writing of commands or data to the device
- at intermediate points in complex routines
- at routine exit points

Note again that you don't have to restrict yourself to a single `xxdebug` variable, or to binary tests that check to see if a variable is on or off. You can use as many variables, and as many values for each variable, as necessary to reflect the functional divisions most appropriate to your driver. It might even be useful to get truly esoteric, and send certain trace statements directly to the user tty (by calling `uprntf()`) while the rest use `printf()` and go to the console.

## Event-Triggered Printing

In the above discussion, the `xxdebug` variable was initialized by the compiler, and toggled with a debugger. However, it's just as easy to have the driver routines themselves set a trigger variable under pre-chosen conditions.

For example, if you wanted to enable tracing after a given *condition* had occurred, you could declare `xxdebug`, just as was shown above, but define `XXDPRINT` somewhat differently:

```
#ifdef XXDEBUG
int xxdebug = 0;
#define XXDPRINT(v,msg,a1,a2) \
    if (xxdebug > (v)) printf(msg,a1,a2);
#endif
```

and then, in the code that checks for the condition:



```
#ifdef XXDEBUG
if (condition) xxdebug = 1;
#endif
```

Then to call XXDPRINT:

```
#ifdef XXDEBUG
XXDPRINT(0, "driver name...\n", a, b);
#endif
```

One major disadvantage of using the kernel `printf()` is that its output doesn't go through a device driver, and thus can't be paused with Control-S or redirected to a file. It's possible, then, that `printf()` will overwhelm you with output. There are a number of things that you can do if you run into this problem:

- If you haven't used multivalued print-control variables, then do so. This gives you more control than you have with simple on/off print control, and will allow you to reduce the amount to trace noise.
- You can use a debugger to set the global variable `noprintf`. This will keep `printf()`'s output from being sent to the console, but that output will still go to a buffer where kernel error messages are kept before being transferred to `/var/adm/messages`. You can examine the message buffer at your leisure, in one of two different ways:
  - From a user window, you can use `dmesg`.
  - From `kadb` (or `adb` on `/dev/kmem`) you can type `msgbuf+10/s`.
- It's also possible to reconfigure your system so that it uses a hardcopy terminal as its console over a RS-232 line. Then, you won't lose any of the `printf()` output.
- Best of all, you can get another machine and connect it to your machine over a RS-232 line. Having done so, use `tip` to open a window on the second machine *as the console of the test machine*. You can then use `tip`'s record feature (see the `tip` man page) to make a record of all the stuff that `printf()` is sending to the test machine's console.

## Asynchronous Tracing

As mentioned above, there are occasions when timing problems forbid the use of the `printf` statement. In these cases, it's a good idea to give up any attachment that you might have to `printf()` statements and use `kadb`.

Or, if you prefer, it's possible to deal with timing problems by using `kadb` to patch the `noprintf` variable, and then to check the message buffer to see what's going on. Doing so:

- allows you to continue using the debugging code that you installed before encountering the timing problem, and

- presents you with a sequential list of the events in your driver, a list spelled out in English phrases and including interrupt-level events.

Or, you can simply use `kadb` for everything.

## `kadb` — A Kernel Debugger

**NOTE** *kadb does not work with versions of the kernel earlier than 3.2.*

`kadb` is an interactive debugger similar in operation to `adb`. `kadb` differs in several key respects from `adb`. It runs as a standalone program under the PROM monitor, rather than as a user process in user address space. And it allows you to set breakpoints and single step in the kernel!

Thus, running a kernel under `kadb` is significantly different than running it under `adb -k`. The `k` option to `adb` merely makes it simulate the kernel memory mappings while `kadb` actually runs in the kernel address space. And unlike `adb`, which runs at user level and as a separate process from the process being debugged, `kadb` runs in system space as a *coprocessor*. It shares not only the kernel address space but its CPU supervisor mode as well.

`kadb`, for all intents and purposes, is a version of `adb`. It has the same command syntax and almost the same command set. Thus, you can see the `kadb` and `adb` manual pages, as well as *Debugging Tools for the Sun Workstation*, for more details on its use. Note, however, the following points of special interest to driver developers:

- All interrupts are disabled while interacting with `kadb` (except non-maskable interrupts). Thus, when using `kadb` to examine memory, the kernel remains stable. However, while single stepped instructions are being executed, the actual standing priority of the kernel is temporarily restored, and interrupts can get dispatched, run and return. You won't notice unless you have a break point set in the interrupt routine, which works just fine.
- `kadb` is installed so that, when a program is being run under it, an abort sequence (L1-A) will transfer control not to the PROM monitor but to `kadb` itself. Once in `kadb`, you can abort again and be transferred to the monitor. The transfer is direct and immediate, so you can use the monitor to examine control spaces (e.g. page and segment maps) which are not accessible from `kadb`. The monitor `c` command will return you to `kadb`.
- `kadb` runs in the same virtual memory space as the kernel itself, and with the CPU in supervisor mode. This means that `kadb` uses the kernel memory maps when calculating virtual addresses, and that it can directly examine kernel structures. This is in contrast to the situation with `adb -k`, where software copies of the page table entries are used to map virtual addresses to physical pages.
- `kadb`'s memory view is almost the same as that resulting from `adb /vmunix /dev/kmem`. In other ways, however, `kadb` is much different. To give just one example: on Sun-3 and Sun-3x machines, where users and supervisors share the virtual address space, `kadb` allows the user to examine the user virtual address space (this is *not* true with `adb -k`).



- Finally, be aware that `kadb` — as a consequence of the way that `adb` works — always does 32-bit memory reads. Even if you tell `kadb` to read a byte it will read a long. This leads to a lack of control that can cause problems when reading device registers. (This problem does not exist on the Sun386i. On the Sun386i, when `kadb` is told to read a byte, it does. Within `kadb`, the `B` command is used to read a single byte and the `v` command to write one).

## 5.4. Device Driver Error Handling

There are various types of errors: “expected” errors like those generated by `xprobe()` routines, transient errors in operations that can reasonably be retried, fatal errors that require controlled shutdowns, and others. The kinds of errors that you will face depends upon the kinds of drivers that you write and the peculiarities of your devices; few generalizations can usefully be made.

To further complicate matters, the detection and treatment of errors varies greatly from device to device. You should begin by carefully reading your device specification manual to determine the error indications that can arise and the responses that should be made when they do. Most devices have at least an error bit in the control/status register, and usually more detailed error information is available. Ideally, you should understand all potential errors, avoid those that you can and recover from the rest. This ideal isn’t always achievable, but try not to leave any obvious holes. *If you do nothing else, check for device errors and use the kernel `printf()` function to report them to the system console.*

There are various error reporting and management mechanisms available to the driver developer. Most of them have already been mentioned as they’ve become relevant; here they are collected and summarized:

### Error Recovery

It’s difficult to generalize about error-recovery mechanisms, for they are largely device specific. It’s worth noting, however, that:

- Some errors are worth retrying and some aren’t; the matter is entirely device specific.
- Error-recovery routines should be able to run at the interrupt level. This is because errors can occur either synchronously or asynchronously with respect to the dispatch of device commands, and, therefore, recovery routines must be callable from interrupt routines.
- If you do implement error recovery logic, you must do so consistently. The data structure that contains retry-status information must be global, and must be protected by critical sections. Error-recovery routines, like interrupt routines in general, must take special pains to protect data-structure integrity; indeed, they must *restore* such integrity upon encountering errors they can’t recover from.

## Error Returns

There are three mechanisms by which driver routines can report errors up to their calling routines. The first, of course, is by way of the values that the driver routines return to their callers. The second, and most important, is the error-reporting mechanism based upon the buffer-header. *This is the only mechanism that can be used when returning errors from `xxstrategy()`, `xxstart()`, and `xxintr()`.* (See the discussion of `xxintr()` error reporting in the *Summary of Device Driver Routines* chapter. Finally, it is possible to directly set the global error register, `u.u_error`, from routines in the top half of the driver.

## Error Signals

It is sometimes desirable to have a driver send a software interrupt to the process or processes. It's possible, for example, that a device will fail in an unrecoverable fashion — in this case it's perhaps a good idea to signal the user processes, rather than merely returning an extraordinary error code. It's also possible (though rare) for a driver to encounter serious errors from which it can recover by restarting the device — user processes may also need to be notified in this case. The kernel `psignal()` and `gsignal()` routines can signal either a single process or all the processes in a given process group.

## Error Logging

When you use the kernel `printf()` statement to report errors to the console, those errors are also placed into a system error-message buffer accessible to the `dmesg` daemon. `dmesg` can be, and typically is, run every 30 minutes by the `crontab` daemon, for the purpose of appending the messages in the buffer to `/var/adm/messages`. Note that the message buffer is small, and that if a lot of entries are being written into it, some of them will get lost before being transferred into `/var/adm/messages`.

## Kernel Panics

The most unequivocal way of dealing with an error is to panic when you get it. The `panic()` routine is provided to help you do so in a somewhat controlled fashion — `panic()` is called only on unresolvable fatal errors. It prints "panic: mesg" on the console, and then reboots. (Or, if you're running under the debugger, it transfers control to `kadb`). `panic()` also keeps track of whether it's already been called, and avoids attempts to sync the disks (by flushing all pending write buffers) if it has, since this can lead to recursive panics.

The final production version of a driver should call `panic()` only when "impossible" situations are encountered; lesser errors should be recovered from. During debugging, though, `panic()` can be used to implement a passable assert mechanism.

```
#ifdef XXDEBUG
if (inconsistent condition)
    panic("Assertion Failed: ...");
#endif
```

(It's possible to write a fancier assert mechanism, for example by having an `ASSERT` macro which calls an `assert()` routine which prints error context information and then calls `panic()`, but this minimal hack will perhaps do).



Finally, note that in cases where it's *very* important to halt the system *immediately* after detecting an inconsistent condition, `kadb` can be used. The driver code can test for the inconsistent condition, and then set a debugging variable:

```
if (inconsistent condition)
    junk = 1;
```

`kadb` can then be used to set a breakpoint at the machine instruction generated from the assignment to `junk`.

## 5.5. System Upgrades

System upgrades generally have minimal effects on user-written device drivers. The changes that are necessary are rare and release specific.

Some changes must be made if user-written drivers are to work with new release software. In Release 2.0, for example, there was a minor change in one of the bus-interface structures. There wasn't much involved in adapting user-written drivers, but it had to be done.

In other cases, changes are optional. When VMEbus machines were introduced, for example, drivers had to be adapted to run on them; however, it was possible to upgrade Multibus machines without rewriting user-written drivers.

In any case, any release upgrades that imply changes — either optional or mandatory — to user-written device drivers will be documented in the *System Summary and Change Notes* for the release in question.

## 5.6. Loadable Drivers

The Sun386i supports loadable drivers. This feature allows you to add a device driver to a running system without rebooting the system or rebuilding the kernel. The loadable drivers feature reduces time spent on driver development, and makes it easier for users to install drivers from other vendors.

This section explains how to convert a non-loadable driver to be a loadable driver.

Conversion of a non-loadable driver to a loadable driver requires an initialization or "wrapper" module to be written. The module `zzinit.c` below is an example of a wrapper module that contains the same kind of information ordinarily provided by a config file and by the linker. Almost all wrappers are identical to the example below. Usually, only the actual structure initialization values are different.

The following module is a typical example of an initialization routine for a driver named `zz` that has one controller and one device on that controller.

```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/buf.h>
#include <sys/param.h>
#include <sys/errno.h>
#include <sundev/mbvar.h>
#include <sun/autoconf.h>
```

```

#include <sun/vddrv.h>

extern zzopen(), nulldev(), zzstrategy(), zzdump();
extern zzsize(), zzread(), zzwrite(), zziocntl();
extern zzint(), nodev(), seltrue();

extern struct mb_driver zzcdriver;  /* defined in driver */

/*
 * Driver block device entry points (normally in <sun/conf.c>)
 */
struct bdevsw zzbdev = {
    zzopen, nulldev, zzstrategy, zzdump, zzsize, 0
};

/*
 * Driver character device entry points (normally in <sun/conf.c>)
 */
struct cdevsw zzcdev = {
    zzopen, nulldev, zzread, zzwrite, zziocntl, nodev,
    nulldev, seltrue, 0
};

/*
 * Controller structure (normally in ioconf.c) (see <sundev/mbvar.h>)
 */
struct mb_ctlr zzctlr[] = {
    &zzcdriver, 0, 0, (caddr_t) 0x00001000, 2, 6,
    SP_ATMEM, 0
};

/*
 * Device structure (normally in ioconf.c) (see <sundev/mbvar.h>)
 */
struct mb_device zzcdevice[] = {
    &zzcdriver, 0, 0, 0, (caddr_t) 0x00000000, 0, 0, 0x0,
    0, 0x0
};

/*
 * The following structure is defined in <sun/vddrv.h>
 *
 * If the number of controllers is 0, then the address of the
 * controller structure array must be NULL. Similarly, if the number
 * of devices is 0, then the address of the device structure array
 * must be NULL. The bdevsw or cdevsw entries can be NULL if there
 * is no block or character device for the driver.
 */
struct vldrv vd = {
    VDMAGIC_DRV,      /* Type of module. This one is a driver. */
    "zzdrv",          /* Name of the module. */
    zzctlr,            /* Address of the mb_ctlr structure array */

```





```

    &zzcdriver,      /* Address of the mb_driver structure */
    zzcdevice,      /* Address of the mb_device structure array */
    1,              /* Number of controllers */
    1,              /* Number of devices */
    &zzbdev,         /* Address of the bdevsw entry */
    &zzcdev,         /* Address of the cdevsw entry */
    0,              /* Block device number. 0 means let system choose. */
    0,              /* Char. device number. 0 means let system choose. */
};

/*
 * This is the driver entry point routine. The name of the default entry point
 * is xxxinit. It can be changed by using the "-entry" command to modload.
 *
 * inputs: function code - VDLOAD, VDUNLOAD, or VDSTAT.
 *         pointer to kernel vddrv structure for this module.
 *         pointer to appropriate vdioc1 structure for this function.
 *         pointer to vdstat structure (for VDSTAT only)
 *
 * return: 0 for success. VDLOAD function must set vdp->vdd_vdtab.
 *         non-zero error code (from errno.h) if error.
 */

xxxinit(function_code, vdp, vdi, vds)
    unsigned int function_code;
    struct vddrv *vdp;
    addr_t vdi;
    struct vdstat *vds;
{
    switch (function_code) {
        case VDLOAD:
            vdp->vdd_vdtab = (struct vdl linkage *)&vd;
            return (0);
        case VDUNLOAD:
            return (unload(vdp, vdi));
        case VDSTAT:
            return (0);
        default:
            return (EIO);
    }
}

static unload(vdp, vdi)
    struct vddrv *vdp;
    struct vdioc1_unload *vdi;
{
    extern struct buf zztab;

    struct buf *dp;

    dp = &zztab;
    if (dp->b_actf) {

```

```

        return(-1); /* The driver still has an active request. */
    }

    /* The driver can do any device shutdown stuff that it needs to do */

    return(0);
}

```

Your driver routines can be placed in the wrapper module if you like. If your driver is big, it is more appropriate to break it into several modules.

If you decide to place your driver in the wrapper module, then the driver can be compiled with the following command line:

```

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
-DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
-DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFFS \
-DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zzinit.c

```

However, if the driver consists of more than one module, then you must use the link editor, `ld(1)`, with the `-r` option to preserve relocation information. For example you might type:

```

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
-DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
-DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFFS \
-DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zzinit.c

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
-DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
-DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFFS \
-DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zz1.c

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
-DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
-DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFFS \
-DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zz2.c

example# ld -r -o zz.o zzinit.o zz1.o zz2.o

```

Thus the object module can be created either by the `cc(1)` command, when the driver resides in one module, or by the `ld(1)` command, when the driver resides in several modules.

In either case the resulting object file (`zzinit.o` or `zz.o`) is a normal COFF file and can then be used with the `modload` command.<sup>1</sup> The driver is just like

<sup>1</sup> "COFF" = Common Object File Format, a UNIX object-file standard to which Sun386i assembler and link-editor output files (normally `a.out`) comply. See `coff(5)`.



any other program, except its text segment starts somewhere in the range  
0xFD000000 to 0xFE000000.



---

# PROM User's Manual Addenda and Errata



The Sun logo, Sun Microsystems, and Sun Workstation are registered trademarks of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, Sun386i, SunInstall, SunOS, SunView, NFS, NeWS, and SPARC are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

SunIPC is a trademark of Sun Microsystems, Inc.

ALM and ALM2 are trademarks of Sun Microsystems, Inc.

DVMA is a trademark of Sun Microsystems, Inc.

Tapemaster is a trademark of Ciprico, Inc.

Copyright © 1988 - 1989 Sun Microsystems, Inc. – Printed in U.S.A.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means – graphic, electronic, or mechanical – including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

The Sun Graphical User Interface was developed by Sun Microsystems Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees.

---

## Contents

|                                                           |    |
|-----------------------------------------------------------|----|
| New Monitor Command Feature .....                         | 1  |
| EEPROM Security Feature .....                             | 1  |
| The Password .....                                        | 2  |
| Changing Security Modes .....                             | 3  |
| EEPROM Layout for PROM Security .....                     | 4  |
| 3-D Logo EEPROM Parameter .....                           | 4  |
| Sun-3 Extended Tests .....                                | 5  |
| New Boot Path Extended Tests .....                        | 5  |
| Sun-3/400 Series and Sun-3/80 PROM .....                  | 5  |
| The Power-Up Test Sequence .....                          | 6  |
| Diagnostic LEDs .....                                     | 7  |
| Sun-3/80 LED .....                                        | 8  |
| The Boot Sequence .....                                   | 8  |
| Diagnostic Power-Up .....                                 | 9  |
| More Interactive Self-Test Commands .....                 | 11 |
| A Successful Sun-3/80 Self-Test .....                     | 11 |
| Test Loop Menu — Sun-3/400 Series and Sun-3/80 .....      | 12 |
| Successful Diagnostic Boot Display .....                  | 12 |
| Remote Testing .....                                      | 13 |
| Diagnostic Self-test Sequence .....                       | 14 |
| Sun-3/80 LED .....                                        | 20 |
| Self-Test Descriptions .....                              | 22 |
| LED Register Test .....                                   | 22 |
| UART SCC (Z8530) Port A,B Write/Read Test .....           | 22 |
| Keyboard/Mouse SCC (Z8530) Port A,B Write/Read Test ..... | 23 |



|                                                        |    |
|--------------------------------------------------------|----|
| System Enable Register Read Test .....                 | 23 |
| PROM Checksum Test .....                               | 24 |
| I/O Mapper Write/Write/Read Test .....                 | 24 |
| I/O Mapper RAM Address Test .....                      | 25 |
| I/O Mapper RAM 3-Pattern Test .....                    | 25 |
| Bus Error Register Test .....                          | 25 |
| Level 1 Interrupt Test .....                           | 26 |
| Level 2 Interrupt Test .....                           | 26 |
| Level 3 Interrupt Test .....                           | 26 |
| TOD Clock Interrupt Test .....                         | 26 |
| Memory Write/Write/Read Test .....                     | 27 |
| Memory Address Test .....                              | 27 |
| Memory 3-Pattern Test .....                            | 27 |
| Memory Read Byte Alignment Test .....                  | 28 |
| Memory Write Byte Alignment Test .....                 | 28 |
| Parity Memory No Error Test .....                      | 29 |
| Parity Memory Forced Error Test .....                  | 30 |
| ECC Memory No Error Test .....                         | 30 |
| ECC Memory Forced CE Test .....                        | 32 |
| ECC Memory Forced UE Test .....                        | 33 |
| Central Cache Tag RAM Write/Write/Read Test .....      | 34 |
| Central Cache Tag RAM Address Test .....               | 34 |
| Central Cache Tag RAM 3-Pattern Test .....             | 35 |
| Central Cache Data RAM Write/Write/Read Test .....     | 35 |
| Central Cache Data RAM Address Test .....              | 36 |
| Central Cache Data RAM 3-Pattern Test .....            | 36 |
| Central Cache Data RAM Read Byte Alignment Test .....  | 36 |
| Central Cache Data RAM Write Byte Alignment Test ..... | 36 |
| Central Cache Read Hit Test .....                      | 37 |
| Central Cache Invalid Read Miss Test .....             | 38 |
| Central Cache Valid Read Miss Test .....               | 39 |
| Central Cache Write Hit Test .....                     | 40 |
| Central Cache Write Miss, No Writeback Test .....      | 41 |

|                                                                     |    |
|---------------------------------------------------------------------|----|
| Central Cache Write Miss, Writeback Test .....                      | 42 |
| Central Cache Line Cross Invalid Read Miss Test .....               | 43 |
| Central Cache Line Cross Write Miss Writeback Test .....            | 44 |
| Central Cache Writeback Timeout Test .....                          | 45 |
| Block Copy (Source=Cache Miss, Dest=Cache Miss) Test .....          | 46 |
| Block Copy (Source=Cache Miss, Dest=Cache Hit) Test .....           | 47 |
| Block Copy (Source=Cache Hit, Dest=Cache Miss) Test .....           | 48 |
| Block Copy (Source=Cache Hit, Dest=Cache Hit) Test .....            | 49 |
| Memory Write/Write/Read Read Test (Central Cache on) .....          | 50 |
| IOC Tag RAM Write/Write/Read Test .....                             | 50 |
| IOC Tag RAM Address Test .....                                      | 51 |
| IOC Tag RAM 3-Pattern Test .....                                    | 51 |
| IOC Data RAM Write/Write/Read Test .....                            | 51 |
| IOC Data RAM Address Test .....                                     | 52 |
| IOC Data RAM 3-Pattern Test .....                                   | 52 |
| IOC Data RAM Read Byte Alignment Test .....                         | 52 |
| IOC Data RAM Write Byte Alignment Test .....                        | 52 |
| VME Loopback Test .....                                             | 53 |
| VME Loopback and DVMA Test .....                                    | 54 |
| IOC Read Hit Test .....                                             | 55 |
| IOC Invalid Read Miss Test .....                                    | 56 |
| IOC Write Hit Test .....                                            | 57 |
| IOC Write Miss, No Writeback Test .....                             | 58 |
| IOC Write Miss, Writeback Test .....                                | 59 |
| IOC Read Miss, Writeback Test .....                                 | 60 |
| IOC Valid Write Hit (Central Cache Match, Unmod) Test .....         | 61 |
| IOC Invalid Read Miss (Central Cache Match, Unmod) Test .....       | 63 |
| IOC Invalid Read Miss (Central Cache Match, Modified)<br>Test ..... | 65 |
| IOC Valid Read Miss (Central Cache Match), Writeback<br>Test .....  | 66 |
| IOC Flush (Valid, Modified) Test .....                              | 68 |
| IOC Flush (Valid, Not Modified) Test .....                          | 69 |



|                                                           |    |
|-----------------------------------------------------------|----|
| IOC Flush (Not Valid, Not Modified) Test .....            | 70 |
| I/O Mapper Invalid Page (IO.DT) Test .....                | 71 |
| IOC Write Miss, Writeback (Write Protect) Test .....      | 72 |
| IOC Invalid Read Miss (IO Mapper IO.EN = 0) Test .....    | 73 |
| IOC Write Miss (IO Mapper IO.EN = 0) Test .....           | 74 |
| IOC Random Data Block Write Test .....                    | 75 |
| IOC Random Data Block Read (Central Cache off) Test ..... | 76 |
| IOC Random Data Block Read (Central Cache on) Test .....  | 77 |
| P4 Overlay Frame Buffer Write/Write/Read Test .....       | 78 |
| P4 Overlay Frame Buffer Address Test .....                | 79 |
| P4 Overlay Frame Buffer 3-Pattern Test .....              | 79 |
| P4 Overlay Frame Buffer March Test .....                  | 80 |
| P4 Overlay Frame Buffer Read Byte Alignment Test .....    | 80 |
| P4 Overlay Frame Buffer Write Byte Alignment Test .....   | 80 |
| P4 Enable Plane Write/Write/Read Test .....               | 81 |
| P4 Color Plane Write/Write/Read Test .....                | 82 |
| Printer Controller Check — Sun-3/80 .....                 | 82 |
| Clock/Calendar Device — 3/80 Only .....                   | 82 |
| LANCE Ethernet Controller Check — 3/80 only .....         | 83 |
| ESP SCSI Check — 3/80 Only) .....                         | 83 |
| Host System Initialization .....                          | 83 |
| The Sun-3 PROM Monitor .....                              | 83 |
| Bringing up the PROM Monitor .....                        | 84 |
| Conventions .....                                         | 84 |
| Monitor Command Overview .....                            | 84 |
| Executing a Command .....                                 | 85 |
| Default Values .....                                      | 86 |
| Word Sizes .....                                          | 86 |
| The Monitor Commands .....                                | 86 |
| Displaying and Modifying Memory .....                     | 86 |
| Special Monitor Commands .....                            | 89 |
| Address Increment/Decrement Command .....                 | 89 |
| The ^r Command — Sun-3/400 series only .....              | 89 |

|                                                                  |     |
|------------------------------------------------------------------|-----|
| The <b>^t</b> Command .....                                      | 89  |
| The <b>^i</b> Command .....                                      | 90  |
| The <b>^c</b> Command .....                                      | 90  |
| The <b>!</b> Command .....                                       | 90  |
| Regular Monitor Commands .....                                   | 90  |
| Monitor <b>a</b> Command .....                                   | 90  |
| Monitor <b>A</b> Command—Sun-3/400 Series .....                  | 91  |
| Monitor <b>b</b> Command .....                                   | 91  |
| Monitor <b>c</b> Command .....                                   | 93  |
| Monitor <b>d</b> Command .....                                   | 93  |
| Monitor <b>e</b> Command .....                                   | 93  |
| Monitor <b>f</b> Command .....                                   | 93  |
| Monitor <b>F</b> Command—Sun-3/400 Series .....                  | 94  |
| Monitor <b>g</b> Command .....                                   | 94  |
| Monitor <b>h</b> Command .....                                   | 94  |
| Monitor <b>i</b> Command — Sun-3/200 or Sun-3/400 series .....   | 94  |
| Monitor <b>j</b> Command — Sun-3/200 or Sun-3/400 Series .....   | 95  |
| Monitor <b>k</b> Command .....                                   | 95  |
| Monitor <b>l</b> Command .....                                   | 95  |
| Monitor <b>m</b> Command .....                                   | 95  |
| Monitor <b>n</b> Command — Sun-3/200 and 3/400 series only ..... | 96  |
| Monitor <b>o</b> Command .....                                   | 96  |
| Monitor <b>p</b> Command .....                                   | 96  |
| Monitor <b>q</b> Command .....                                   | 96  |
| Monitor <b>r</b> Command .....                                   | 97  |
| Monitor <b>R</b> Command—(Sun-3/400 Series only) .....           | 98  |
| Monitor <b>s</b> Command .....                                   | 98  |
| Monitor <b>T</b> Command—Sun-3/400 Series .....                  | 98  |
| Monitor <b>u</b> Command .....                                   | 99  |
| Monitor <b>v</b> Command .....                                   | 101 |
| Monitor <b>w</b> Command .....                                   | 101 |
| Monitor <b>x</b> Command .....                                   | 101 |
| Monitor <b>y</b> Command — Sun-3/200 or -3/400 Series .....      | 101 |



|                                                            |     |
|------------------------------------------------------------|-----|
| 1.1. SPARCsystem 330 Self-Tests and Monitor Commands ..... | 103 |
| Self-Test Interaction .....                                | 103 |
| Successful Self-Test Display .....                         | 107 |
| To Read the CPU Board LED Table .....                      | 107 |
| The SPARCsystem 330 PROM Monitor .....                     | 109 |
| Monitor <b>z</b> Command .....                             | 110 |
| Monitor <b>s</b> Command .....                             | 110 |
| Monitor <b>t</b> Command .....                             | 110 |
| Monitor <b>v</b> Command .....                             | 110 |
| Monitor <b>^a</b> Command .....                            | 111 |
| Monitor <b>^c</b> Command .....                            | 111 |
| Monitor <b>!</b> Command .....                             | 111 |
| Identifying A Faulty Memory Module .....                   | 111 |
| SPARCsystem 330 Extended Tests .....                       | 111 |
| SPARCsystem 330 Initialization .....                       | 113 |

---

## Tables

|                                                          |     |
|----------------------------------------------------------|-----|
| Table 1-1 Self-Test Execution Time Comparisons .....     | 7   |
| Table 1-2 Miscellaneous Registers for the 68020 .....    | 97  |
| Table 1-3 Miscellaneous Registers for the 68030 .....    | 97  |
| Table 1-4 Function Code Values .....                     | 98  |
| Table 1-5 Port Arguments .....                           | 100 |
| Table 1-6 Option Arguments .....                         | 100 |
| Table 1-7 SPARCsystem CPU Board LED Interpretation ..... | 108 |





---

## Figures

|                                                                                    |     |
|------------------------------------------------------------------------------------|-----|
| Figure 1-1 Sun-3/75, 3/140, 3/150, 3/160, and 3/110 Diagnostic Boot Sequence ..... | 14  |
| Figure 1-2 Sun-3/50 and Sun-3/60 Diagnostic Boot Sequence .....                    | 15  |
| Figure 1-3 Sun-3/260 and Sun-3/280 Diagnostic Boot Sequence .....                  | 16  |
| Figure 1-4 Sun-3/400 Series Diagnostic Boot Sequence .....                         | 17  |
| Figure 1-5 Sun-3/80 Diagnostic Boot Sequence .....                                 | 19  |
| Figure 1-6 Sun-3 Monitor Help Menu .....                                           | 85  |
| Figure 1-7 SPARCsystem 330 Diagnostic Boot-Up Display .....                        | 104 |
| Figure 1-8 Diagnostic Boot-Up Display - Continued .....                            | 105 |
| Figure 1-9 Diagnostic Boot-Up Display - Continued .....                            | 106 |
| Figure 1-10 Ethernet Loopback Connector .....                                      | 112 |





---

# PROM User's Manual Addenda and Errata

The *PROM User's Manual* describes PROM monitor commands, self-tests and extended tests for Sun-2 through Sun-386i systems. This text adds to or modifies the information found there, for Sun-3 systems with the new, 2.8 version of the Boot PROM, and all Sun-3/400 series, the Sun-3/80 and SPARCsystem 330s.

## New Monitor Command Feature

In the past, all numerical PROM monitor commands were entered in hexadecimal. If you have PROM version 2.8, or a Sun-3/400 series system, a Sun-3/80 or SPARCsystem 330, you may now enter decimal or ASCII values after the PROM monitor prompt ( > ). This feature is particularly useful when using the monitor *q* command to program the EEPROM, which sometimes requires that you convert letters and decimal numbers to hexadecimal values before you enter them.

To enter a decimal value after a PROM monitor command, simply precede the value with the “%” character:

```
>q 050 %20
```

To enter an ASCII character, simply precede it with the “@” character:

```
>q 022 @i  
>q 023 @e
```

If the value you enter is not preceded by a % or @ character, the monitor program treats that the value as hexadecimal.

## EEPROM Security Feature

Chapter 10 of the *PROM User's Manual* describes the Sun-3 and Sun-4 EEPROM. There is now a Security Mode Select Feature, located at EEPROM address 0x492. This feature provides a *non-secure* mode that permits the use of all PROM monitor commands.

It also provides a *command secure* mode that permits the use of PROM monitor commands (other than *c*, for continue, or *b*, for boot, without parameters) only when a password is entered. In the command secure mode, you may operate your workstation normally, including powering down, booting, terminating with the L1-A command, and re-booting. You may not, however, perform any unusual operations, such as booting a non-standard kernel, running diagnostics,



or changing EEPROM or CPU board memory contents, without entering a password.

Finally, this feature provides a *fully secure* mode that does not permit use of the PROM monitor (other than the **c** command with no parameters) without entering a password. To power-up, re-boot, or perform any other PROM monitor operation, you must supply a password. This mode allows you to control access to the workstation by turning it off. The workstation does not automatically boot on power-up.

**CAUTION** In fully secure mode, even a default boot cannot be completed unless the password is entered. Once the SunOS is halted, you cannot restore it until you enter the correct password after the prompt. If the password is unknown, the system CPU board must be serviced as a failed board.

### The Password

If you should attempt to enter a PROM monitor command such as **q**, for example, on a system that is set for one of the secure modes, your interaction might look like this:

```
>q 020                (you want to look at EEPROM offset address 020)
>Mon Pass:            (you enter the correct password)
>EEPROM 020: 00 ?     (the contents of location 020 are shown)
```

Or, if you enter an incorrect password, your interaction might look like this:

```
>q 020
>Mon Pass: (you enter the wrong password)
>Invalid
                (there is a slight delay)
>
```

You may now try again or enter an unprotected command.

To install or change a password, the system must be in non-secure mode, or you must know the existing password for a secure system. You then use the PROM monitor **q** command to enter the password in EEPROM offset location 490 for Sun-3 and Sun-4 (SPARC) systems, or 160 for the Sun386i.

**NOTE** When you attempt to change the values stored in the EEPROM monitor password locations, you will be prompted with this message:

```
Modifying security location(s). Are you sure?(y/N)
```

If you enter **y** for yes, the change you entered is written to the location shown. If you enter **n** for no, nothing is written to EEPROM, and the contents of the next location are displayed.

To enter a monitor password, use the **@** character, described at the beginning of this document, in order to enter the letters that make up the password. If you do not use the **@** character, you must enter the hexadecimal equivalent of the letters. Following is an example of the way you would enter a password called *mypasswd* on a Sun-3 or Sun-4 system. You enter one letter per location, followed with : **[Return]**. To exit the command, you may enter any non-



hexadecimal character, such as period, as shown.

```
>q 493
> EEPROM 493: 00? @m
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 494: 00? @y
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 495: 00? @p
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 496: 00? @a
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 497: 00? @s
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 498: 00? @s
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 499: 00? @w
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 49a: 00? @d
Modifying security location(s). Are you sure? (y/N) Yes
> EEPROM 49b: 00? .
>
```

**NOTE** *If a password was already stored in locations 493-49a, hexadecimal values would appear in place of the zeroes in the example above.*

*The password you enter must either fill the eight bytes (locations 493-49a) with a character or a zero.*

*Note that changes to the security mode and password do not take effect until the PROM monitor mode is re-entered.*

It is recommended that the password is changed before the security mode is changed. For more information on using the EEPROM **q** command, refer to the *PROM User's Manual*, Sun PN 800-1736, or the Monitor(8S) section of the *SunOS Reference Manual*.

## Changing Security Modes

The EEPROM offset location 492 (or 162 for a Sun386i) contains a value that determines the security mode. The table below shows the interpretation of values found in that location. The "0x" denotes a hexadecimal value.

|                  |                |
|------------------|----------------|
| 0x1              | command secure |
| 0x5e             | fully secure   |
| all other values | non-secure     |

Because the PROM monitor password is stored as text, it is recommended that the `chmod` and `/etc/chown` commands be used so that the `/dev/eepprom` device file may be accessed by the super-user. To accomplish this, enter:



```
%su
Password: enter your super-user password
# cd /dev
# /etc/chown root eeeprom
# chmod 600 eeeprom
```

### EEPROM Layout for PROM Security

Here is a table that shows the EEPROM offset locations for the Sun-3, Sun-4 and Sun386i systems.

| <i>Sun386i<br/>Offset</i> | <i>Sun-3,-4<br/>Offset</i> | <i>Field</i> | <i>Function</i>                                                                                                                           |
|---------------------------|----------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| 0x160-1                   | 0x490-1                    | bad_login    | The bad login counter stores the number of invalid password attempts. The maximum value is 65535 and the counter does not roll over to 0. |
| 0x162                     | 0x492                      | secure       | The values 1 and 0x5e correspond to command secure and fully secure, respectively. Any other value is non-secure.                         |
| 0x163-a                   | 0x493-a                    | password     | If the password is shorter than 8 bytes, the password string is fenced with a null character.                                             |

### 3-D Logo EEPROM Parameter

If your system has a CG6 board, you may set EEPROM location 0x020 to 0x06 so that, upon power-up, the Sun logo appears to be three-dimensional. Refer to the *PROM User's Manual* monitor **q** command description for information on changing EEPROM values.

## Sun-3 Extended Tests

Chapter 9 of the *PROM User's Manual* describes Sun-3 extended tests. For workstations with the boot PROM version 2.8, many of these tests are unavailable, and the user interface has changed. The new PROM version tests only the devices needed to boot the operating system. Therefore, when you invoke the extended tests from the monitor prompt,

```
>x
```

the menu of extended tests look something like this:

```
Extended Test Menu: (Enter 'q' to exit)

Cmd - Test

ie - Intel Ethernet Test
mk - Mouse/Keyboard Ports Test
rs - Serial Ports Test
```

**NOTE** For CPU boards with the AMD AM7990 (Lance) Ethernet chip, the first choice will be

```
ae - AMD Ethernet Test
```

## New Boot Path Extended Tests

In order to invoke the disk and tape bootpath tests when a version 2.8 PROM is installed on a Sun-3 CPU board, you must enter an asterisk after the boot command, from the monitor prompt:

```
>b*device ()
```

The extended test appropriate for the named *device* will then be executed, and any error messages displayed on the screen. *device* could be one of the following:

```
sd    for SCSI disk
st    for SCSI tape
xd    for Xylogics 7053 Disk Controller
xt    for Xylogics tape
xy    for Xylogics 450/451 Disk Controller
```

## Sun-3/400 Series and Sun-3/80 PROM

For Sun-3/400 series and Sun-3/80 users, the text that follows is intended to replace Chapters 7 and 8 in the *PROM User's Manual*, Sun PN 800-1736-10. These chapters cover changes to PROM monitor commands and self-tests associated with the Sun-3/400 product. In addition, the information in Chapter 13 of the PROM manual, "Sun-4 Extended Test System", applies to the Sun-3/400 series workstation. If you have a Sun-3/400 series system, Chapter 9, "Sun-3 Extended Test System" does not apply.



## The Power-Up Test Sequence

In order to perform the power-up tests, two assumptions must be met. The CPU (the non-PMMU portion of the MC68030 for the Sun-3/400 series system) must be functional and the ability to fetch instructions from the Boot PROM must be intact.

Powering up a Sun-3 workstation resets the CPU to *boot* state, which means that all instruction fetches are forced to the Boot PROMs. Execution of the minimum-confidence power-up tests begin immediately. These tests do not employ any memory until memory has been successfully checked.

The objective of the power-up test sequence is to determine whether or not the CPU board logic and main memory are functional. Following the successful completion of the power-up tests and subsequent workstation initialization, an attempt is made to boot the SunOS operating system, an EEPROM-specified program, or an operator-specified stand-alone program.

If hardware problems are detected during the verification process, the PROM monitor prompt should appear.

For Sun-3/400 series systems, if the Diagnostic Switch at the rear of the system is in NORM position, you will not be able to interact with the self-tests. You may read the LEDs on the CPU board edge (described later in this chapter) to determine whether or not a test is failing, and you will see a rotating diagonal symbol after the

```
Testing _ megabytes of memory...
```

message on the console during the memory tests. The quantity of memory checked during a power-up with the diagnostic switch on NORM is dependent on EEPROM programming. The EEPROM chapter in the *PROM User's Manual* and the `eeeprom` command description in the *SunOS Reference Manual* explain how to set the parameter that controls the quantity of memory tested.

For the Sun-3/80, which has no diagnostic switch, an EEPROM parameter must be set in order to execute a diagnostic boot-up and view the self-test display on a terminal. Refer to the "Diagnostic Power-Up" section for more information on this setting.

If the workstation contains a large amount of main memory, self-tests may last as long as eight minutes. The table below compares the self-test duration of various Sun-3 systems.



Table 1 *Self-Test Execution Time Comparisons*

| <i>System Type</i> | <i>Clock Rate<br/>in MHz</i> | <i>Memory Size<br/>in Megabytes</i> | <i>Self-test Duration<br/>in Minutes</i> |
|--------------------|------------------------------|-------------------------------------|------------------------------------------|
| Sun-3/160          | 16 MHz                       | 4 Meg                               | .5 Min                                   |
| Sun-3/75           | 16 MHz                       | 8 Meg                               | .75 Min                                  |
| Sun-3/60           | 20 MHz                       | 24 Meg                              | 1.75 Min                                 |
| Sun-3/80           | 16 MHz                       |                                     | 11 Sec (est.)                            |
| Sun-3/260          | 25MHz                        | 64 Meg                              | 2.5 Min                                  |
| Sun-3/400 series   | 33MHz                        | 128 Meg                             | 2.75 Min (est.)                          |
| SPARC system       |                              | 128 Meg                             | 8.0 Min (est.)                           |

If the Diagnostic Switch is in the NORM position, (or the Sun-3/80 EEPROM parameter set), the power-up tests execute successfully and, if you do *not* terminate the default boot sequence, an attempt is made to down-load the SunOS operating system.

A display something like this appears on the *workstation's* screen to indicate that power-up tests are successful:

Selftest Completed Successfully.

Sun Workstation, Model Sun-3/\_\_\_\_ Series

Type-X keyboard



ROM Rev \_\_, \_\_ MB memory installed, Serial # \_\_\_\_\_

Ethernet address \_\_:\_\_:\_\_:\_\_:\_\_:\_\_

Testing \_\_ megabytes of memory...Completed.

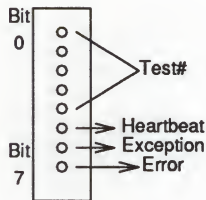
Auto-boot in progress...

## Diagnostic LEDs

One requirement of Sun-3 firmware is to assign a unique test number to most of the power-up tests and display that number in bits *zero* through *four* of the diagnostic LEDs as the test is running. Given that there are fewer test numbers than there are power-up tests, power-up tests that check the same part of hardware share a test number.

If one of these power-up tests should fail, bit *seven* of the diagnostic LEDs also lights up. Bit seven serves as an indicator that there is a hardware problem. The LED display permits the service person not only to conclude whether or not there is a problem, but to determine which type of power-up test is failing.





For the sake of completeness, LED *five* is the *heart beat* LED. After the power-up tests have been completed, but prior to invocation of the SunOS operating system or an EEPROM-specified program, LED 5 will blink on and off to indicate that the IU is actually executing instructions. LED *six* indicates whether or not the failure is an exception (i.e. unexpected trap or unexpected interrupt). The diagram to the left shows LED designations for Sun-3/400 series systems.

#### Sun-3/80 LED

Sun-3/80 systems have one green LED that blinks while a test is in progress, and turns OFF when there is an error condition. When the light stays ON, everything is functioning satisfactorily:

| LED status | Visual | Condition |
|------------|--------|-----------|
| Blinking   | ⊕      | Testing   |
| OFF        | ○      | Error     |
| ON (Green) | ●      | Ok        |

#### The Boot Sequence

Following the initialization of the workstation, the default boot sequence is executed. There are two issues that must be considered here. One has to do with *what is to be down-loaded* while the other has to do with *where it is to be loaded from*.

Assuming *no* operator intervention, the position of the Diagnostic Switch (on all Sun-3's except the Sun-3/80) will determine *what* is to be booted. If the Diagnostic Switch is in the NORM position, the SunOS operating system is booted. Otherwise, if the Diagnostic Switch is in the DIAG position, the EEPROM-specified program is booted. Be aware that the PROM monitor program is invoked if no EEPROM-specified program is available.

If you are in the PROM monitor mode, you may specify *what* is to be booted and *where* it is to be booted from. See command **b** (boot) in the chapter titled *Sun-3 PROM Monitor Commands* for a description of how to boot user-specified programs from user-specified devices.

If you are interacting with a Sun-3 workstation through the console to reach the monitor program, you may enter L1-a (or **Break** on a terminal) IMMEDIATELY after the

Testing \_\_ megabytes of memory...Completed.

message.

#### CAUTION

**Do not use L1-A procedure once the automatic boot has started; the file systems may be damaged if disks are powered-on and the operating system has started to run.**

If the operating system is already booted, use the procedures described in Chapter 3 of the *PROM User's Manual* to start the monitor.

Once you are in the monitor mode (symbolized by the **>** prompt), you should do the following:

```
> q 0
panic: zero
Syncing disks... done
```

Press L1-a or **Break** again when the message above finishes.

Next you may see this message:

```
dumping to dev somevalue, offset somevalue
Abort at somevalue
>
```

The firmware also determines from what boot device the program will be loaded. If the Diagnostic Switch is in the NORM position and the content of EEPROM location 0x18 is equal to 0x12 (an arbitrarily chosen value), Sun-3 firmware will attempt to boot the SunOS operating system from the boot path specified in the EEPROM, beginning at location 0x19. If the boot path is missing or contains an error, the monitor program is invoked.

If the Diagnostic Switch is in the NORM position and the content of EEPROM location 0x18 is *not* equal to 0x12, Sun-3 firmware attempts to boot the SunOS operating system using the following boot device polling sequence:

1. Xylogics Disk (450-451).
2. SCSI Disk.
3. Ethernet.

If the Diagnostic Switch is in the DIAG position, the firmware assumes that both the path name of the file containing the to-be-loaded program and the boot device are specified in the EEPROM, beginning at EEPROM location 0x22. If either the file name or the boot device is not present or is in error, the monitor is invoked. If the Diagnostic Switch is in the DIAG position, you may connect a terminal to Serial A or B and interact with the self-tests and the Extended Test System, if required.

## Diagnostic Power-Up

**NOTE** *There is no diagnostic switch on a Sun-3/80; an EEPROM setting is required to allow a diagnostic boot-up that displays test names and errors through Serial Port A. Enter the PROM monitor mode and use the **q** command to write **12** to location 0x70b. Any other value causes tests to run without error reporting to the terminal. Refer to "Displaying and Modifying Memory" for information on use of the **q** command.*

If the Diagnostic Switch exists and is on DIAG, the self-test is executed as it is when the switch is "off" or on NORM, except that all of memory is tested. In addition, self-test status information is directed only to serial ports A and B, using the MMU (Memory Management Unit) bypass until all hardware required for the Video Monitor has been successfully tested.



Any hardware failures during the selftests will invoke scope loops to permit troubleshooting the failure. An RS-232 terminal with its characteristics set to 9600 Baud, 8 data bits, 1 stop bit and no parity should be connected to Serial Port A of the CPU board if you wish to view self-test status and interact with the system. If you use Serial Port B you must set the terminal baud rate to 1200.

Limited interaction with the self-test program is possible in this mode and the following characters will invoke the following actions when entered from the terminal connected to serial port A during self test. Each of these commands is documented below.

#### ESCAPE Key — Sun-3/400 Series and Sun-3/80

Pressing this key any time during the self-test sequence prior to the display of the `Selftest Passed` message causes the self-test sequence to abort and a warning message is displayed. The necessary memory sizing and other setup is done, and then, for the Sun-3/400 series, the program displays the PROM Monitor menu. For the Sun-3/80, this message is displayed:

```
<Warning: selftests aborted by user>
```

```
<Initializing Main Memory 0x0000000C Megabytes Initialized>
```

```
Type a Character within 10 seconds to enter the Menu Tests... (a for e mode)
```

```
EEPROM: Using RS232 A port.
```

```
Selftest Completed.
```

Note that this abort can be done if the test is running normally (without any errors) or if the test is presently looping on any error encountered.

#### Control-q Key — Sun-3/400 Series Only

Holding down the `(Control)` key while pressing the `q` key any time during the execution of a particular Sun-3/400 Series self-test causes that one self-test to stop and execution of the next self-test to begin. For example, if the `(Control-q)` sequence was entered during the `MEMORY WRITE/WRITE/READ TEST`, that test would terminate and the next test (`MEMORY ADDRESS TEST`) would begin. Note that this action can be performed if the test is running normally (without any errors) or if the test is presently looping on any error encountered.

#### Control-l Key — Sun-3/400 Series and Sun-3/80

Holding down the `(Control)` key while pressing the `l` key any time during a self-test causes that particular self-test to terminate and control to be transferred to the `TEST LOOP MENU`. From within this menu any particular self-test can be executed and looped directly, without executing the self-tests that normally come before it. Once a test is invoked from the `TEST LOOP MENU`, a test control flag is set so that once that test has finished it will go back to the beginning of that one particular test and start it all over again. In order to proceed to the next self-test after a particular self-test has been invoked from this menu, enter the `(Control-q)` sequence or the `(Esc)` key to go to next test or to the Monitor (as described above). See the section on the Test Loop Menu for more details.

## More Interactive Self-Test Commands

- b** Press the **b** (a mnemonic for *burn-in*) key, prior to the display of the ...Completed or Selftest Finished message, to execute the power-up test sequence indefinitely. This option is useful during the manufacturing burn-in stage.

For the Sun-3/80, when the last selftest is finished the message testsAutomaticallycontinuing will be displayed and the self-test will be restarted again, using the System Enable Register read test as the first test. The SCC and terminal I/O tests are bypassed in burn-in mode since operator interaction is required. This looping sequence will continue until an ESCAPE is entered, a reset is done, or the "b" key pressed again to turn burn-in mode off. The burn-in mode can be toggled by successive pressing of the "b" key. Note that this burn-in mode key can be processed during a test that is running normally (no errors), if the test is presently looping on an error encountered, or at the end of selftest when the "Selftest Finished" is displayed and an operator input is requested.

- s** Press the **s** key prior to the display of the ...Completed message to *re-start* the power-up test sequence.

### Space Bar

If one of the power-up tests fails, it will continue to re-execute forever unless interrupted. Press the **[space bar]** to terminate the failed test and execute the next power-up test.

### Control-m

Holding down the **[Control]** key while pressing **m** causes this test to end and sets a flag that allows execution of only the quick memory tests.

### Control-b

Holding down the **[Control]** key while entering **b** causes this test to end and sets a test control flag so that all the tests that require the Bus Error circuitry to work will be skipped. This test is also the equivalent of a "Control-m". It assumes that 8MB of ECC memory are present on the board, because the test is executed before memory has been sized. This test displays no error messages. The Bus Error dependent tests skipped are:

- Bus Error Register Test
- The Level 1 to Level 3 Interrupt tests
- P4 Video RAM Board Tests.

## A Successful Sun-3/80 Self-Test

For a Sun-3/80, After the selftest have completed the final status of the selftest sequence will be displayed, as shown by the following example:



```
END OF SELFTEST # 0x00000005 (SELFTEST FAILED)
#PASSED = 0x00000004, #FAILED = 0x00000001
```

```
<Initializing Main Memory 0x0000000C Megabytes Initialized>
```

```
Type a Character within 10 seconds to enter the Menu Tests... (e for echo mode)
```

### Test Loop Menu — Sun-3/400 Series and Sun-3/80

The Test Loop Menu is a menu appears when you press **[Control-L]** (loop) key either during Sun-3/400 series or Sun-3/80 self-test execution. Test. The menu allows the operator to transfer self-test control directly to a specified self-test and loop that test continually. This option is intended to support debugging a specific failing test or section of hardware without running all of the previous self-tests.

Once in the Test Loop Menu the following display will be shown on the terminal:

```
<<< TEST LOOP MENU >>>
(a) System Enable Register Test
(b) I/O Mapper RAM Write/Write/Read Test
(b) I/O Mapper RAM Address Test
(c) I/O Mapper RAM 3-Pattern Test
(t) ECC Memory Forced CE Test
(u) ECC Memory Forced UE Test
<<< Press ,Space for more, <Esc> to quit or select an option >>>
```

*or, for the Sun-3/80*

```
<<< TEST LOOP MENU >>>
(a) System Enable Register Test
(b) I/O Mapper RAM Write/Write/Read Test
(b) I/O Mapper RAM Address Test
(c) I/O Mapper RAM 3-Pattern Test
(w) P4 Enable plane Write/Write Read Test
(x) P4 Color Plane Write/Write/Read Test
<<< Press ,Space for more, <Esc> to quit or select an option >>>
```

At this point you may either press the space bar to see more self-tests, or enter the key code for the test you want to execute in loop mode. Once a test is selected, that test is invoked and executed continually, regardless of whether the test passed or failed. From this point you can use any of the special control keys to continue the selftest if desired: **[control-q]** to continue the self-test in non-loop mode or **[Esc]** to abort all self-tests and go to the PROM Monitor.

### Successful Diagnostic Boot Display

For a Sun-3/400 series or Sun-3/80 system, upon self-test completion a status message that looks something like this is displayed:

```
END OF SELFTEST #0x00000005 (SELFTEST FAILED)

#PASSED=0x00000004, #FAILED=0x00000001
```



**NOTE** *There is no diagnostic switch on a Sun-3/80; an EEPROM setting is required to allow a diagnostic boot-up. Enter the PROM monitor mode and use the **q** command to write **12** to location 0x70b. Refer to "Displaying and Modifying Memory" for information on use of the **q** command.*

*During a diagnostic boot, after self-test has completed successfully, you will be prompted to enter the extended tests, as shown below.*

Selftest passed.

Optional Menu Tests

Type a character within 10 seconds to enter Menu Tests... (e for echo mode)

If you press a terminal key during this time the Extended Test Menu is displayed on the terminal screen. See the "Sun-3 Extended Test Sequence" chapter in this document for details.

If you do not assert control of the system by pressing a terminal key within the 10 seconds delay time the Boot PROM program will next display the Sun logo and message on the console.

## Remote Testing

If you press **e** on a *dumb terminal* keyboard, during the ten second period, all subsequent output will appear on *both* the console video monitor *and* a terminal attached to Serial Port A or B.

The purpose of this feature is to enable an individual at a remote site, using a terminal attached to a non-local machine by way of a telephone line with modems at each end, and the individual on-site to observe the system's behavior simultaneously. Once a system is in this mode, the local and non-local parties can communicate by employing the **#** command. Specifically, if the person at the remote site would like to send a message to the person on-site, he or she would simply type the **#** before typing the actual message. In order to terminate the message, the person at the remote site would enter a second **#**. At this point, the on-site person would be able to respond by prefixing and terminating their response with the **#**.

The **#** option will be very useful in the situation where an on-site customer has contacted a remote repair depot regarding a potential hardware problem. The repair person at the remote repair depot could initiate diagnostics on the non-local machine and allow both parties to observe the output. Beyond that, the repair person and the customer could communicate by way of the **#** command.



## Diagnostic Self-test Sequence

The following text lists diagnostic mode self-tests for each Sun-3 system. If the diagnostic switch is enabled, the name of each test appears on the terminal until all self-tests are complete. The tests differ slightly according to the system architecture. The examples that follow represent each Sun-3 workstation.

Figure 1 *Sun-3/75, 3/140, 3/150, 3/160, and 3/110 Diagnostic Boot Sequence*

Boot PROM Selftest

```
PROM Checksum Test
DVMA Reg Test
Context Reg Test
Segment Map Wr/Rd Test
Segment Map Address Test
Page Map Test
Memory Path Data Test
NXM Bus Error Test
Interrupt Test
TOD Clock Interrupt Test
MMU Access Bit Test
MMU Access/Modify Bit Test
MMU Invalid Page Test
MMU Protected Page Test
Parity Test
Memory Size = 0x00000xxx Megabytes
Memory Test (testing xxxxxxxx MBytes)
```

Selftest passed

Optional Menu Tests

Type character within 10 seconds to enter menu tests... (e for echo mode)



Figure 2 *Sun-3/50 and Sun-3/60 Diagnostic Boot Sequence*

## Boot PROM Selftest

PROM Checksum Test  
Context Reg Test  
Segment Map Wr/Rd Test  
Segment Map Address Test  
Page Map Test  
Memory Path Data Test  
NXM Bus Error Test  
Interrupt Test  
TOD Clock Interrupt Test  
MMU Access Bit Test  
MMU Access/Modify Bit Test  
MMU Invalid Page Test  
MMU Protected Page Test  
Parity Tests  
Memory Size = 0x00000xxx Megabytes  
Memory Test (testing xxxxxxxx MBytes)

Selftest passed

## Optional Menu Tests

Type a character within 10 seconds to enter Menu Tests... (e for echo mode)



Figure 3 *Sun-3/260 and Sun-3/280 Diagnostic Boot Sequence*

Boot PROM Selftest

PROM Checksum Test  
DVMA Reg Test  
Context Reg Test  
Segment Map Wr/Rd Test  
Segment Map Address Test  
Page Map Test  
Memory Path Data Test  
NXM Bus Error Test  
Interrupt Test  
TOD Clock Interrupt Test  
MMU Access Bit Test  
MMU Access/Modify Bit Test  
MMU Invalid Page Test  
MMU Protected Page Test  
ECC Error Tests  
Cache Data 3 Pat Test  
Cache Tags 3 Pat Test  
Memory Size = 0x00000xxx Megabytes  
Memory Test (testing xxxxxxxx MBytes)

Selftest passed

Optional Menu Tests

Type character in next 10 seconds to enter menu tests... (e for echo mode)



Figure 4 *Sun-3/400 Series Diagnostic Boot Sequence*

```

System_name Boot Prom Self-Test
(Hit key for character echo, space bar for next test, control-m, control-b)
System Enable Register Read Test {pass 0x00000001}
PROM Checksum Test
I/O Mapper RAM Write/Write/Read Test {pass 0x00000001}
I/O Mapper RAM Address Test {pass 0x00000001}
I/O Mapper RAM 3-Pattern Test {pass 0x00000001}
Bus Error Register Test {pass 0x00000001}
Level 1 Interrupt Test {pass 0x00000001}
Level 2 Interrupt Test {pass 0x00000001}
Level 3 Interrupt Test {pass 0x00000001}
TOD Clock Interrupt Test {pass 0x00000001}
<Sizing Main Memory>
<Memory Size = 0x00000010>
Memory Write/Write/Read Test {pass 0x00000001}
Memory Address Test {pass 0x00000001}
Memory 3-Pattern Test {pass 0x00000001}
Memory Read Byte Alignment Test {pass 0x00000001}
Memory Write Byte Alignment Test {pass 0x00000001}
Parity Memory No Error Test {pass 0x00000001}
Parity Memory Forced Error Test {pass 0x00000001}
ECC Memory No Error Test {pass 0x00000001}
ECC Memory Forced CE Test {pass 0x00000001}
ECC Memory Forced UE Test {pass 0x00000001}
Central Cache Tag RAM Write/Write/Read Test {pass 0x00000001}
Central Cache Tag RAM Address Test {pass 0x00000001}
Central Cache Tag RAM 3-Pattern Test {pass 0x00000001}
Central Cache Data RAM Write/Write/Read Test {pass 0x00000001}
Central Cache Data RAM Address Test {pass 0x00000001}
Central Cache Data RAM 3-Pattern Test {pass 0x00000001}
Central Cache Data RAM Read Byte Alignment Test {pass 0x00000001}
Central Cache Data RAM Write Byte Alignment Test {pass 0x00000001}
Central Cache Read Hit Test {pass 0x00000001}
Central Cache Invalid Read Miss Test {pass 0x00000001}
Central Cache Valid Read Miss Test {pass 0x00000001}
Central Cache Write Hit Test {pass 0x00000001}
Central Cache Write Miss, No Writeback Test {pass 0x00000001}
Cache Write Miss, Writeback Test {pass 0x00000001}
Central Cache Line Cross Invalid Read Miss Test {pass 0x00000001}
Central Cache Line Cross Write Miss Writeback Test {pass 0x00000001}
Central Cache Writeback Timeout Test {pass 0x00000001}
Block Copy (Source=Cache Miss, Dest=Cache Miss) Test {pass 0x00000001}
Block Copy (Source=Cache Miss, Dest=Cache Hit) Test {pass 0x00000001}
Block Copy (Source=Cache Hit, Dest=Cache Miss) Test {pass 0x00000001}
Block Copy (Source=Cache Hit, Dest=Cache Hit) Test {pass 0x00000001}

```



```

Memory Write/Write/Read Test (Central Cache on) {pass 0x00000001}
IOC Tag RAM Write/Write/Read Test {pass 0x00000001}
IOC Tag RAM Address Test {pass 0x00000001}
IOC Tag RAM 3-Pattern Test {pass 0x00000001}
IOC Data RAM Write/Write/Read Test {pass 0x00000001}
IOC Data RAM Address Test {pass 0x00000001}
IOC Data RAM 3-Pattern Test {pass 0x00000001}
IOC Data RAM Read Byte Alignment Test {pass 0x00000001}
IOC Data RAM Write Byte Alignment Test {pass 0x00000001}
VME Loopback Test {pass 0x00000001}
VME Loopback and DVMA Test {pass 0x00000001} (not for Sun-3/460)
IOC Read Hit Test {pass 0x00000001}
IOC Invalid Read Miss Test {pass 0x00000001}
IOC Write Hit Test {pass 0x00000001}
IOC Write Miss, No Writeback Test {pass 0x00000001}
IOC Write Miss, Writeback Test {pass 0x00000001}
IOC Read Miss, Writeback Test {pass 0x00000001}
IOC Valid Write Hit (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Write Miss (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Read Miss (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Read Miss (Central Cache Match,Modified) Test {pass 0x00000001}
IOC Valid Read Miss (Central Cache Match),Writeback Test {pass 0x00000001}
IOC Flush (Valid, Modified) Test {pass 0x00000001}
IOC Flush (Valid, Not Modified) Test {pass 0x00000001}
IOC Flush (Not Valid, Not Modified) Test {pass 0x00000001}
IO Mapper Invalid Page (IO.DT) Test {pass 0x00000001}
IOC Write Miss, Writeback (Write Protect) Test {pass 0x00000001}
IOC Invalid Read Miss (IO Mapper IO.EN = 0) Test {pass 0x00000001}
IOC Write Miss (IO Mapper IO.EN = 0) Test {pass 0x00000001}
IOC Random Data Block Write Test {pass 0x00000001}
IOC Random Data Block Read (Central Cache off) Test {pass 0x00000001}
IOC Random Data Block Read (Central Cache on) Test {pass 0x00000001}
<P4 Low Resolution Color Video RAM Board Detected>
P4 Overlay Frame Buffer Write/Write/Read Test {pass 0x00000001}
P4 Overlay Frame Buffer Address Test {pass 0x00000001}
P4 Overlay Frame Buffer 3-Pattern Test {pass 0x00000001}
P4 Overlay Frame Buffer March Test {pass 0x00000001}
P4 Overlay Frame Buffer Read Byte Alignment Test {pass 0x00000001}
P4 Overlay Frame Buffer Write Byte Alignment Test {pass 0x00000001}
P4 Enable Plane Write/Write/Read Test {pass 0x00000001}
P4 Color Plane Write/Write/Read Test {pass 0x00000001}

```

Selftest passed

Type character within 10 seconds to enter Extended Tests System (e for echo mode)



Figure 5 Sun-3/80 Diagnostic Boot Sequence

```

Sun-3/80 Boot PROM Selftest Revision xx
(Press <Esc> to abort tests or <Ctrl-L> for the Loop Menu)
System Enable Register Read Test {pass 0x00000001} hh:mm:ss
PROM Checksum Test {pass 0x00000001} hh:mm:ss
Clock/Calendar Device {pass 0x00000001} hh:mm:ss
I/O Mapper RAM Write/Write/Read Test {pass 0x00000001} hh:mm:ss
I/O Mapper RAM Address Test {pass 0x00000001} hh:mm:ss
I/O Mapper RAM 3-Pattern Test {pass 0x00000001} hh:mm:ss
<Memory Size = 0x00000010> hh:mm:ss
Memory Address Test {pass 0x00000001} hh:mm:ss
Memory Read Byte Alignment Test {pass 0x00000001} hh:mm:ss
Memory Write Byte Alignment Test {pass 0x00000001} hh:mm:ss
I/O Mapper RAM Write/Write/Read Test {pass 0x00000001} hh:mm:ss
I/O Mapper RAM Address Test {pass 0x00000001} hh:mm:ss
I/O Mapper RAM 3-Pattern Test {pass 0x00000001} hh:mm:ss
Bus Error Register Test {pass 0x00000001} hh:mm:ss
Level 1 Interrupt Test {pass 0x00000001} hh:mm:ss
Level 2 Interrupt Test {pass 0x00000001} hh:mm:ss
Level 3 Interrupt Test {pass 0x00000001} hh:mm:ss
Configuration Memory Check {pass 0x00000001} hh:mm:ss
LANCE Controller Check {pass 0x00000001} hh:mm:ss
EPS SCSI Check {pass 0x00000001} hh:mm:ss
Floppy Controller Check {pass 0x00000001} hh:mm:ss
Printer Controller Check {pass 0x00000001} hh:mm:ss
<P4 Low Resolution Color Video RAM Board Detected>
P4 Overlay Frame Buffer Write/Write/Read Test {pass 0x00000001} hh:mm:ss
P4 Overlay Frame Buffer Address Test {pass 0x00000001} hh:mm:ss
P4 Overlay Frame Buffer 3-Pattern Test {pass 0x00000001} hh:mm:ss
P4 Overlay Frame Buffer March Test {pass 0x00000001} hh:mm:ss
P4 Overlay Frame Buffer Read Byte Alignment Test {pass 0x00000001} hh:mm:ss
P4 Overlay Frame Buffer Write Byte Alignment Test {pass 0x00000001} hh:mm:ss
P4 Enable Plane Write/Write/Read Test {pass 0x00000001} hh:mm:ss
P4 Color Plane Write/Write/Read Test {pass 0x00000001} hh:mm:ss

END OF SELFTEST #0x00000005 (SELFTEST PASSED/FAILED)
#PASSED = 0x00000004, #FAILED = 0x00000001

Initializing Main Memory n Megabytes Initialized

Type a character within 10 seconds to enter menu tests.

```

Refer to the *PROM User's Manual* for descriptions of the self-tests named for workstations other than the Sun-3/400 series and the Sun-3/80.

Here is a summary of the LED displays for the Sun-3/400 series and Sun-3/80 during the power-up self-tests. Many of the tests are the same for both systems; the test description will indicate which tests is system-specific. In case of error in a Sun-3/400 series test, the LED in bit position 7 also lights. The text that follows this chart shows both the normal test and error displays. As indicated, many tests share the same LED display.



## Sun-3/80 LED

| LED status | Visual | Condition |
|------------|--------|-----------|
| Blinking   | ⊖      | Testing   |
| OFF        | ○      | Error     |
| ON (Green) | ●      | Ok        |

The Sun-3/80 has one green LED only, which lights steadily during normal function, blinks during a test, and turns off when there is an error. The messages that appear on a terminal attached to Serial Port A describe the specific test in progress and the nature of any errors found.

The following table explains the Sun-3/400 series LED code.

| <b>LED Display</b><br>● = ON, ○ = OFF<br>7 6 5 4 3 2 1 0 | <b>SelfTest being Performed.</b>               |
|----------------------------------------------------------|------------------------------------------------|
| ● ● ● ● ● ● ● ●                                          | A reset will set LEDs to this state.           |
| ○ ○ ○ ○ ○ ○ ○ ●                                          | Keyboard/Mouse SCC Write/Read Test             |
| ○ ○ ○ ○ ○ ○ ● ●                                          | System Enable Register Read Test               |
| ○ ○ ○ ○ ○ ● ○ ○                                          | PROM Checksum Test                             |
| ○ ○ ○ ○ ○ ● ○ ●                                          | I/O Mapper RAM Test(s)                         |
| ○ ○ ○ ○ ○ ● ● ○                                          | Bus Error Register Test                        |
| ○ ○ ○ ○ ○ ● ● ●                                          | Interrupt Test(s)                              |
| ○ ○ ○ ○ ● ○ ○ ○                                          | ECC Memory Sizing and Test(s)                  |
| ○ ○ ○ ○ ● ○ ○ ●                                          | Parity Memory Sizing and Test(s)               |
| ○ ○ ○ ○ ● ○ ● ○                                          | ECC Memory Forced Error Test(s)                |
| ○ ○ ○ ○ ● ○ ● ●                                          | Central Cache Tag RAM Test(s)                  |
| ○ ○ ○ ○ ● ● ○ ○                                          | Central Cache Data RAM Test(s)                 |
| ○ ○ ○ ○ ● ● ○ ●                                          | Central Cache Hit/Miss Test(s)                 |
| ○ ○ ○ ○ ● ● ● ○                                          | Block Copy Test(s)                             |
| ○ ○ ○ ○ ● ● ● ●                                          | Memory Write/Write/Read Test(Central Cache on) |
| ○ ○ ○ ● ○ ○ ○ ○                                          | IOC Tag RAM Test(s)                            |
| ○ ○ ○ ● ○ ○ ○ ●                                          | IOC Data RAM Test(s)                           |
| ○ ○ ○ ● ○ ○ ● ○                                          | VME Loopback Test                              |
| ○ ○ ○ ● ○ ○ ● ●                                          | VME Loopback and DVMA (not for 3/460)          |
| ○ ○ ○ ● ○ ● ○ ○                                          | IOC Read/Write/Flush Test(s)                   |
| ○ ○ ○ ● ○ ● ○ ●                                          | P4 Overlay Frame Buffer Test(s)                |



## Self-Test Descriptions

The following paragraphs describe each individual test, and the messages and indications generated if it fails. Some tests do not apply to both the Sun-3/400 series and Sun-3/80; refer to the Diagnostic Boot Sequence display examples on previous pages for lists of applicable self-tests.

### LED Register Test

The first Sun-3/400 series test is indicated by a slow loop through the Diagnostic LEDs, from LED 7 through LED 0. It is intended to determine if the CPU is able to fetch instructions correctly from the Diagnostic PROM and transfer data across the data bus to the LED status register.

**NOTE** *If all the LED s on a Sun-3/400 series remain lighted, you could have a low voltage or board seating problem, or the system could contain the wrong Boot PROM.*

**NOTE** *The diagrams that follow depict the LED states for each self-test. Note that the LEDs are shown here for convenience in reading as binary numbers; the least significant bit is on the right. In most systems, the LEDs are read with bit 0 on the left for desktop installations and with bit 0 on top for deskside installations.*

The tests are described here in the order that they are executed.

### UART SCC (Z8530) Port A,B Write/Read Test

This test checks the ability of the Sun-3/400 series workstation CPU to communicate with port A and B of the Zilog Z8530 Serial Communications Chip (SCC). It performs a write/read test of port A and B SCC chip internal registers WR12/RR12. The purpose of this test is to test the path between the CPU and the SCC so that all subsequent tests may display the test name and error status to a terminal attached to Serial Port A.

The test enters a scope loop on a data compare error, but no error messages are displayed on the terminal during this test.

The diagram below summarizes the LED states for a Sun-3/400 series CPU board.

| <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|----------------------------------|------------------------------|------------------|
| 0x00                             | bit 7 00000000 bit 0         | okay             |
| 0x80                             | ●0000000                     | error            |

### Keyboard/Mouse SCC (Z8530) Port A,B Write/Read Test

This test checks the ability of the CPU to communicate with port A and B of the Zilog Z8530 Serial Communications Chip (SCC). It performs a write/read test of port A and B SCC chip internal registers WR12/RR12. Note that this test is for the SCC used for the keyboard and mouse.

The test enters a scope loop on a data compare error, but no error messages are displayed on the terminal during this test because this is a test of the path to the SCC chip.

The diagram below summarizes the LED states for a Sun-3/400 series workstation.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 1                  | 0x01                             | bit 7 ○○○○○○● bit 0          | okay             |
| 1                  | 0x81                             | ●○○○○○●                      | error            |

### System Enable Register Read Test

This test reads the System Enable Register and verifies that all bits read are zero, ignoring the Diagnostics Switch bit.

The test enters a scope loop on data compare errors, with the following terminal message:

```
exp xxxxxxxx, obs xxxxxxxx, xor xxxxxxxx
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 3                  | 0x03                             | bit 7 ○○○○○○● bit 0          | okay             |
| 3                  | 0x83                             | ●○○○○○●                      | error            |



## PROM Checksum Test

The checksum of all locations in both PROMs with the exception of the last word is calculated and compared to an expected value that is stored in the last word of the second PROM. If this test fails, the PROMs should be replaced.

The test enters a scope loop on data compare errors, with the following terminal message:

Checksum error: Exp xxxxxxxx, Obs xxxxxxxx

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 4           | 0x04                      | bit 7 ○○○○○●○○ bit 0  | okay      |
| 4           | 0x84                      | ●○○○○●○○              | error     |

## I/O Mapper Write/Write/Read Test

This test verifies the address and data paths to the IO Mapper RAM, as well as address bit and data bit uniqueness.

For each test address of IO Mapper RAM:

(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+iomapper\_size)

For each data pattern at each test address:

(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)

The test does the following:

- Writes test data to the test address.
- Writes inverted test data to test address + 0x04.
- Reads back data from the test address and compares.

Upon error, the test loops through the steps shown above with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following terminal message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 5           | 0x05                      | bit 7 ○○○○○●● bit 0   | okay      |
| 5           | 0x85                      | ●○○○○●●               | error     |

**I/O Mapper RAM Address Test**

This test writes the complete I/O Mapper RAM address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Mapper RAM.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display shown for the I/O Mapper Write/Write/Read Test applies to this test also.

**I/O Mapper RAM 3-Pattern Test**

This test writes the complete I/O Mapper RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display shown for the I/O Mapper Write/Write/Read Test applies to this test also.

**Bus Error Register Test**

This test does the following:

1. Verifies that attempting to read an invalid memory address causes a bus error, with appropriate data written to the BUSERR register.
2. Verifies that attempting to read an FPA device address while the ENABLE-FPA bit in the System Enable Register is off causes a bus error, with appropriate data written to the BUSERR register.

This test enters a scope loop upon error, with one or more of the following terminal error messages:

```
error1: No Bus Error Reading Invalid Memory Address, read addr = 0xyyyyyyyy.
error2: Bus error reg TIMEOUT bit not set.
error3: No Bus Error Reading Disabled FPA, read addr = 0xyyyyyyyy.
error4: Bus error reg FPAENERR bit not set.
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 6                  | 0x06                             | bit 7 ○○○○○● bit 0           | okay             |
| 6                  | 0x86                             | ●○○○○●○                      | error            |



**Level 1 Interrupt Test**

This test forces a Level 1 soft interrupt to verify that an autovector Level 1 interrupt will occur.

This test enters a scope loop upon error, with the following error message:

error: No level 1 interrupt occurred.

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 7                  | 0x07                             | bit 7 ○○○○○●● bit 0          | okay             |
| 7                  | 0x87                             | ●○○○○●●                      | error            |

**Level 2 Interrupt Test**

This test forces a level 2 soft interrupt to verify that an autovector Level 1 interrupt will occur.

The test enters a scope loop upon error, with the following terminal error message:

error: No level 2 interrupt occurred.

The LED display for this test is the same as that for the Level 1 Interrupt test.

**Level 3 Interrupt Test**

This test forces a level 3 soft interrupt to verify that an autovector Level 3 interrupt will occur.

The test enters a scope loop upon error, with the following error message:

error: No level 3 interrupt occurred.

The LED display for this test is the same as that for the Level 1 Interrupt test.

**TOD Clock Interrupt Test**

This test enables the TOD CLock to interrupt and then verifies that the interrupt occurs.

This test enters a scope loop upon error, with the following error message:

error: No TOD interrupt occurred.

The LED display for this test is the same as that for the Level 1 Interrupt test.

## Memory Write/Write/Read Test

This test verifies the address and data paths to Main Memory, as well as address bit and data bit uniqueness.

For each test address of main memory:  
(0x0,0x1,0x2,0x4,0x8,0x10,0x20,...size\_of\_mem\_installed)

For each data pattern at each test address:  
(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)

The test does the following:

- Writes test data to test address.
- Writes inverted test data to test address + 0x04.
- Reads back data from test address and compares.

Upon error, the test loops through the steps shown above with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 8           | 0x08                      | bit 7 ○○○○●○○○ bit 0  | okay      |
| 8           | 0x88                      | ●○○○●○○○              | error     |

## Memory Address Test

This test writes the complete Main Memory address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the Main Memory RAM.

This test enters a scope loop on data compare errors, with the following terminal message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display is that same as that shown for the Memory Write/Write/Read Test.

## Memory 3-Pattern Test

This test writes the complete Main Memory address space with a repeated three long word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following terminal message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display is that same as that shown for the Memory Write/Write/Read Test.



### Memory Read Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

The test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

### Memory Write Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

**Parity Memory No Error Test**

This test will execute only if a Parity Memory board was detected when main memory was sized.

The test does the following:

1. Determines if Parity memory is installed. If not, it bypasses the test.
2. Initializes all Parity memory with zero to initialize the check bits.
3. For each test address 0x00000000, 0x00000001, 0x00000002, ..., 0x00040000, 0x00800000, 0x00800001, 0x00800002, ..., 0x00840000, or up to total Parity memory installed (16MB max):

For each data pattern 0x00000001, 0x00000003, 0x00000007, ..., 0xffffffff at one specific test address:

The test:

- (a) Writes 0x0 to Memory Error Address register to clear previous interrupt.
- (b) Enables Parity checking on Parity board.
- (c) Enables Level 7 interrupts in Memory Error Control Register.
- (d) Enables system interrupts in System Interrupt Register.
- (e) Verifies that Memory Error Control register was properly set.
- (f) Writes longword test data to test address.
- (g) Reads longword from test address.
- (h) Verifies that no Level 7 interrupt occurred.
- (i) Verifies that Memory Error Control register was not modified.

Upon error, the test loops through steps a - i with a constant test address and constant test data.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
error2: Unexpected parity error (level 7) interrupt:
testaddr=0x00000000, testdata=0x00000000
Memory Error Control Reg: obs=0x00000000
Parity Memory Error Reg:  obs=0x00000000
Memory Error Address Reg: obs=0x00000000
error3: Bad Memory Error Control Reg after memory write:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 9                  | 0x09                             | bit 7 ○○○○●○○● bit 0         | okay             |
| 9                  | 0x89                             | ●○○○●○○●                     | error            |



### Parity Memory Forced Error Test

This test will execute only if a Parity Memory board was detected when main memory was sized.

For each test address 0x00000000,0x00000001,0x00000002,...,0x00040000, 0x00800000,0x00800001,0x00800002,...0x00840000 or up to total Parity memory installed (16MB max):

For each data pattern 0x1,0x2,0x4,...0x00000100 at one specific test address:

The test does the following:

- Writes 0x0 to Memory Error Address register to clear previous interrupt.
- Enables Parity memory board.
- Writes longword 0x0 to test address, test address + 4 (good parity).
- Enables Parity checking and Parity Test on Parity board.
- Enables Level 7 interrupts in Memory Error Control Register.
- Enables system interrupts in system interrupt Register.
- Writes longword test data to test address.
- Reads longword from test address.
- Verifies that Level 7 interrupt occurred.
- Verifies that Memory Error Address register captured test address.
- Verifies that Parity Memory Error register was set correctly
- Verifies that Memory Error Control register was set correctly.

Upon error, the test loops through the steps shown above with a constant test address and constant test data.

Possible error messages for this test are:

error1: no parity error (level 7) interrupt occurred when bad parity forced.  
addr = 0x00000000, wr/rd data = 0x00000000

error2: Memory Error Address Reg: exp=0x00000000, obs=0x00000000  
addr = 0x00000000, wr/rd data = 0x00000000

error3: Parity Memory Error Reg: exp=0x00000000, obs=0x00000000  
addr = 0x00000000, wr/rd data = 0x00000000

error4: Bad Memory Error Control Reg after memory write:  
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000

The LED display shown for the Parity Memory No Error Test applies to this test also.

### ECC Memory No Error Test

This test does the following:

- Checks if ECC memory is installed. If not, it bypasses this test.
- Determines the starting address and size of total ECC memory.
- Initializes all ECC memory with zero to initialize ECC check bits.
- For each test address 0x00000000,0x00000001,0x00000002,...,0x00040000, 0x00800000,0x00800001,0x00800002,...0x00840000, 0x01000000,0x01000001,0x01000002,...0x01040000, 0x01800000,0x01800001,0x01800002,...0x01840000, ...,

0x08800000,0x08800001,0x08800002,...0x08800000, or up to total memory installed (144MB max) and for each data pattern 0x00000001,0x00000003,0x00000007,...0xffffffff at one specific test address, the test does the following:

- (a) Enables ECC checking on all ECC boards.
- (b) Enables Level 7 interrupts in the Memory Error Control Reg.
- (c) Enables system interrupts in the System Interrupt Register.
- (d) Verify that Memory Error Control register was properly set.
- (e) Reads a longword from test address.
- (f) Verifies that no Level 7 interrupt occurred.
- (g) Verifies that the Memory Error Control register was not modified.

Upon error, the test loops through steps a - g with a constant test address and constant test data.

Possible error messages for this test are:

error1: Memory Error Control Reg not written:

exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000

error2: Unexpected ECC error (level 7) interrupt:

testaddr=0x00000000, testdata=0x00000000

Memory Error Control Reg: obs=0x00000000

Memory Error Address Reg: obs=0x00000000

Syndrome Reg (Board 0): obs=0x00000000

error3: Bad Memory Error Control Reg after memory write:

exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 10                 | 0x0a                             | bit 7●●●●●●●●bit 0           | okay             |
| 10                 | 0x8a                             | ●●●●●●●●                     | error            |



## ECC Memory Forced CE Test

This test does the following:

- Gets the starting address and size of total ECC memory.
- For each test address 0x00000000,0x00800000,0x01000000,...0x07800000, (starting at ECC memory base), or up to total ECC memory installed, the test:
  - (a) Resets syndrome register.
  - (b) Enables CE, level 7 interrupts in the Memory Error Control Register.
  - (c) Verifies that Memory Error Control register was properly set.
  - (d) Writes data to memory.
  - (e) Writes check bits to the Diagnostic registers.
  - (f) Enables ECC checking, DM1 on memory board.
  - (g) Enables system interrupts in the System Interrupt Register.
  - (h) Reads a longword from test address.
  - (i) Verifies that a Level 7 interrupt occurred.
  - (j) Verifies that the Memory Error Control register was modified correctly.
  - (k) Verifies that the CE bit was set in the Memory board syndrome register.
  - (l) Verifies that the syndrome code in the syndrome register was correctly set.

Upon error, the test loops through steps a - l.

Possible error messages for this test are:

error1: Memory Error Control Reg not written:  
exp=0x00000000, obs=0x00000000, testaddr=0x00000000

error2: No ECC error (level 7) interrupt occurred after CE forced:  
testaddr=0x00000000

error3: Bad Memory Error Control Reg after CE forced:  
exp=0x00000000, obs=0x00000000, testaddr=0x00000000

error4: CE bit in ECC syndrome reg not set after CE forced:  
testaddr=0x00000000

error5: Bad ECC Syndrome Reg Syndrome code (b31:24) after CE forced:  
testaddr=0x00000000, exp=0x00000000, obs 0x00000000

The LED display for this test is the same as that for the ECC Memory No Error Test.

**ECC Memory Forced UE Test**

This test does the following:

- Determines the starting address and size of total ECC memory.
- For each test address 0x00000000,0x00800000,0x01000000,...0x07800000, (starting at ECC memory base), or up to total ECC memory installed, the test:
  - (a) Resets the Syndrome Register.
  - (b) Enables Level 7 interrupts in the Memory Error Control Register.
  - (c) Verifies that the Memory Error Control register was properly set.
  - (d) Writes data to memory.
  - (e) Writes check bits to the Diagnostic registers.
  - (f) Enables ECC checking, DM1 on the memory board.
  - (g) Enables system interrupts in the System Interrupt register.
  - (h) Reads a longword from test address.
  - (i) Verifies that a Level 7 interrupt occurred.
  - (j) Verifies that the Memory Error Control register was correctly modified.
  - (k) Verify that the syndrome code in Syndrome register was correctly set.

Upon error, the test loops through steps a -k.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000.
```

```
error2: No ECC error (level 7) interrupt occurred after UE forced:
testaddr=0x00000000
```

```
error3: Bad Memory Error Control Reg after UE forced:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000
```

```
error5: Bad ECC Syndrome Reg Syndrome code (b31:24) after UE forced:
testaddr=0x00000000, exp=0x00000000, obs 0x00000000
```

The LED display for this test is the same as that for the ECC Memory No Error Test.



### Central Cache Tag RAM Write/Write/Read Test

This test verifies the address and data paths to the Central Cache Tag RAM, as well as address bit and data bit uniqueness.

For each test address of Central Cache TAG RAM:  
(base+0x0,base+0x10,base+0x40,base+0x80,...,base+cache\_size)

For each data pattern at each test address:  
(0x00000000,0x00010000,0x00020000,0x00040000,...0x80000000)

The test does the following:

- (a) Writes test data to test address.
- (b) Writes inverted test data to test address + 0x10 (next tag).
- (c) Reads back data from test address and compare.

NOTE: Only bits 16:31 of data read back are valid!

Upon error, the test loops through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following terminal message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 11                 | 0x0b                             | blt 7 ●●●●●●● blt 0          | okay             |
| 11                 | 0x8b                             | ●●●●●●●                      | error            |

### Central Cache Tag RAM Address Test

This test writes the complete Central Cache Tag RAM address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. Only the most significant 16 bits (b31:16) are verified, as the least significant 16 (b15:00) are invalid for reads. This is a test for addressing uniqueness of the Central Cache Tag RAM.

This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display for this test is the same as that for the Central Cache Tag RAM Write/Write/Read Test.

**Central Cache Tag RAM 3-  
Pattern Test**

This test writes the complete Central Cache Tag RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data. Only the most significant 16 bits (b31:16) are verified, because the least significant 16 (b15:00) are invalid for reads.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that for the Central Cache Tag RAM Write/Write/Read Test.

**Central Cache Data RAM  
Write/Write/Read Test**

This test verifies the address and data paths to the Central Cache Data RAM, as well as address bit and data bit uniqueness.

For each test address of Central Cache Data RAM:

(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+cache\_size)

For each data pattern at each test address:

(0x0,0x1,0x2,0x4,0x10,0x20.....0x80000000)

The test does the following:

- (a) Writes test data to test address.
- (b) Writes inverted test data to test address + 0x04.
- (c) Reads back data from test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 11                 | 0x0b                             | bit 7 ○○○○●●● bit 0          | okay             |
| 11                 | 0x8b                             | ●○○○●●●                      | error            |



### Central Cache Data RAM Address Test

This test writes the complete Central Cache Data RAM address space with the longword address as data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the Central Cache Data RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 12                 | 0x0c                             | bit 7 ○○○○●○○ bit 0          | okay             |
| 12                 | 0x8c                             | ●○○○●○○                      | error            |

### Central Cache Data RAM 3- Pattern Test

This test writes the complete Central Cache Data RAM address space with a repeated three-long- word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

### Central Cache Data RAM Read Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

The test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

### Central Cache Data RAM Write Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

**Central Cache Read Hit Test**

This test verifies that a data operand read from system memory with the memory block address in the cache valid causes a read from the cache and not from system memory.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOPCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4, ..., 0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes 00-0f as data into memory for cache line of interest.
  - (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
  - (c) Writes test address, valid bit ON in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Reads from test address.
  - (f) Turns off central cache.
  - (g) Verifies that the data read is from cache, not system memory.
  - (h) Verifies cache tag is still correct (unmodified).
  - (i) Verifies cache data is still correct (unmodified).

Upon error, the test loops through steps a - i with a constant test address.

Possible error messages for this test are:

error1: Bad read data on cache valid read:

testaddr=0x00000000, readaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad cache tag ram after cache valid read:

testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad cache data ram after cache valid read:

testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 13                 | 0x0d                             | bit 7 ○○○○●●● bit 0          | okay             |
| 13                 | 0x8d                             | ●○○○●●●                      | error            |



## Central Cache Invalid Read Miss Test

This test verifies that doing a data operand read from system memory with the memory block address in the cache invalid causes a read from system memory and not the cache. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes 00-0f as data into memory for cache line of interest.
  - (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
  - (c) Writes test address, valid bit OFF in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Reads from test address.
  - (f) Turns off central cache.
  - (g) Verifies that the data read is from system memory, not cache.
  - (h) Verifies cache tag is updated correctly.
  - (i) Verifies cache data is updated correctly.

Upon error, the test loop through steps a - i with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000, readaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad cache tag ram after cache invalid read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad cache data ram after cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display for this test is the same as that shown for the Central Cache Read Hit Test.

### Central Cache Valid Read Miss Test

This test verifies that a data operand read from system memory with a valid modulo 64 Kbyte memory block address in the cache causes a read from system memory and not the cache.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes 00-0f as data into the memory line for test address (x).
  - (b) Writes 50-5f as data into the memory line for addr x+64KB.
  - (c) Writes ff-f0 as data into the cache data RAM for addr x.
  - (d) Writes the test address, valid bit ON in cache tag RAM.
  - (e) Turns on central cache.
  - (f) Reads from address x+64KB.
  - (g) Turns off central cache.
  - (h) Verifies that the data read is from memory, not cache.
  - (i) Verifies that the cache tag is updated correctly
  - (j) Verifies that cache data is updated correctly.

Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad cache tag ram after cache valid write hit:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad cache data ram after cache valid write hit:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the Central Cache Read Hit Test.



### Central Cache Write Hit Test

This test verifies that doing a data operand write to system memory with the memory block address in the cache valid causes a write to the cache and not to system memory.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the longword test address as data into memory.
  - (b) Writes longword zero into cache data RAM.
  - (c) Writes the address, valid bit ON in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Writes the test address with inverted longword address as data.
  - (f) Turns off central cache.
  - (g) Verifies that the system memory location was unmodified, since the write should have gone only to the cache (no write-through).
  - (h) Verifies that the cache tag was updated correctly (valid,dirty,addr bits).
  - (i) Verifies that cache data was written correctly.

Upon error, the test loops through steps a - i with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error2: Bad cache tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error3: Bad cache data ram after cache valid write hit:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

### Central Cache Write Miss, No Writeback Test

This test verifies that a data operand write to system memory address (X+64KB) with a valid, not dirty memory block address (X) in the cache causes a write to the cache and not to system memory, and that no writeback to system memory occurs.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the longword address as data into memory.
  - (b) Writes the inverted longword address as data into cache data RAM.
  - (c) Writes the correct address, valid bit ON in cache tag ram.
  - (d) Turns on central cache.
  - (e) Writes to (address+64KB) with the inverted longword address as data.
  - (f) Turns off central cache.
  - (g) Verifies that the system memory location was unmodified, since the write should have gone only to the cache (no write-through).
  - (h) Verifies that the cache tag was updated correctly (valid,dirty,addr bits).
  - (i) Verifies that cache data was written correctly.

Upon error: loop through steps (a) --> (i) with constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error2: Bad cache tag ram after cache valid write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error3: Bad cache data ram after cache valid write miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.



### Central Cache Write Miss, Writeback Test

This test verifies that a data operand write to system memory address (X+64KB) with a valid and dirty memory block address (X) in the cache causes a write to the cache and also a writeback to system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the longword address as data into memory.
  - (b) Writes inverted longword address as data into cache data RAM.
  - (c) Writes address, valid bit ON, dirty bit ON in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Writes to (address+64KB) with inverted longword address as data.
  - (f) Turns off central cache.
  - (g) Verifies that system memory location was modified, i.e. a writeback of data initially in cache ram occurred.
  - (h) Verifies cache tag was updated correctly (valid, dirty, addr bits).
  - (i) Verifies cache data was written correctly.

Upon error, the test loops through steps a - i with a constant test address.

Possible error messages for this test are:

```
error1: Bad read data on line cross cache invalid read:
testaddr=0x00000000, readaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad cache tag ram after line cross cache invalid read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad cache data ram after line cross cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

## Central Cache Line Cross Invalid Read Miss Test

This test verifies that doing a longword data operand read from system memory across a 16-byte line boundary with invalid memory block addresses in the cache causes a read from system memory and not the cache, and that both 16-byte lines are copied into the cache. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the longword address as data into memory for the two 16-byte lines starting at the address of interest (8 longword writes).
  - (b) Writes the inverted longword address as data into cache data RAM for the two 16-byte lines starting at the address of interest (8 longword writes).
  - (c) Writes the two correct addresses, valid bit OFF in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Reads a longword from (address + 0x0e), to access data across the line boundary.
  - (f) Turns off central cache.
  - (g) Verifies that the data read is the same as the address, and therefore, is not read from cache, but rather from system memory.
  - (h) Verifies that the cache tag is updated correctly (both entries).
  - (i) Verifies that cache data is updated correctly (all 32 bytes).

Upon error, the test loops through steps a - i with a constant test address.

Possible error messages for this test are:

```
error1: Bad read data on line cross cache invalid read:
testaddr=0x00000000, readaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad cache tag ram after line cross cache invalid read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad cache data ram after line cross cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.



### Central Cache Line Cross Write Miss Writeback Test

This test Verifies that a longword data operand write to system memory (X+64KB) across a 16-byte line boundary with a valid and dirty memory block address (X) in the cache causes a write to the cache with the new data and a writeback to memory with the old data in the cache. It verifies that both 16-byte lines are correct in memory as well as cache. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4...0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the longword address as data into memory for the two 16-byte lines at address X and for two lines at address (X+64KB).
  - (b) Writes the inverted longword address as data into cache data RAM for the two 16-byte lines starting at the address of interest.
  - (c) Writes the two correct addresses, valid bit ON and dirty bit ON in cache tag RAM.
  - (d) Turns on central cache.
  - (e) Writes a longword to (address+64KB+0x0e), to access data across the line boundary and to cause a writeback.
  - (f) Turns off central cache.
  - (g) Verifies that memory was modified through writeback of data previously in cache (all 32 bytes).
  - (h) Verifies cache tag is updated correctly (both entries).
  - (i) Verifies cache data is updated correctly (all 32 bytes).

Upon error: loop through steps (a) --> (i) with constant test address. Possible error messages for this test are:

```
error1: Bad memory data on line cross cache write miss writeback:
testaddr=0x00000000, memmaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad cache tag ram after line cross cache write miss writeback:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad cache data ram after line cross cache write miss writeback:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

### Central Cache Writeback Timeout Test

This test verifies that a data operand write to system memory address 0x0 with a valid and dirty memory block address 0x20000000 (a nonexistent memory address) in the cache causes a write to the cache and a writeback timeout due to attempting to writeback to a nonexistent memory address. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4, ..., 0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Write line for addr x (test addr) with 00-0f data.
  - (b) Write cache data ram line for addr x with ff-f0 data.
  - (c) Write address (x+0x9000000), VALID, DIRTY in cache tag ram.  
Maximum addressable memory for Sun-3/400 is 144 MB = 0x9000000.
  - (d) Turn on central cache.
  - (e) Write to address x with longword 0x50515253 as data.
  - (f) Turn off central cache.
  - (g) Verify WBTIMOUT bit Memory Error Control Reg was set.
  - (h) Verify Memory Error Addr Reg captured invalid writeback addr.
  - (i) Verify that entire line at addr x was not modified.
  - (j) Verify cache tag was updated correctly (valid, dirty, addr bits).
  - (k) Verify cache data was written correctly.

Upon error: loop continuously through steps a - k. Possible error messages for this test are:

```
error1: No level 7 interrupt occurred.
testaddr=0x00000000, tagramaddr=0x00000000, dataramaddr=0x00000000
```

```
error2: Memory Error Control Reg Asynch Timeout bit not set.
testaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad Mem Err Addr Reg after cache invalid writeback:
testaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.



### Block Copy (Source=Cache Miss, Dest=Cache Miss) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with all Central Cache entries invalidated causes the proper blocks of data to be transferred in memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOPCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the line for addr x (test addr) in memory with 00-0f data.
  - (b) Writes the line for addr x+0x10 in memory with ff-f0 data.
  - (c) Writes the line for addr x+0x20 in memory with all zero data.
  - (d) Does a block copy read from addr x.
  - (e) Does a block copy write to addr x+0x10.
  - (f) Verifies that the line data for addr x was unmodified.
  - (g) Verifies that the line data for addr x+0x10 was modified correctly.
  - (h) Verifies that the line data for addr x+0x20 was unmodified.

Upon error, the test loops through steps a - h with a constant test address.

Possible error messages for this test are:

error1: Bad system memory where no write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad system memory where block write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad system memory where no write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 13                 | 0x0d                             | bit 7 ●●●●●●● bit 0          | okay             |
| 13                 | 0x8d                             | ●●●●●●●                      | error            |

### Block Copy (Source=Cache Miss, Dest=Cache Hit) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with a valid and clean Central Cache entry for the destination causes the proper blocks of data to be transferred in memory and the Central Cache block entry invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x1, 0x2, 0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the line for addr x (test addr) in memory with 00-0f data.
  - (b) Writes the line for addr x+0x10 in memory with ff-f0 data.
  - (c) Writes the line for addr x+0x20 in memory with all zero data.
  - (d) Writes the Central Cache line for addr x+0x10 with 50-5f data.
  - (e) Writes the Central Cache tag for addr x+0x10 valid, clean.
  - (f) Turns on Central Cache.
  - (g) Does a block copy read from addr x.
  - (h) Does a block copy write to addr x+0x10.
  - (i) Turns off Central Cache.
  - (j) Verifies that the line data for addr x was unmodified.
  - (k) Verifies that the line data for addr x+0x10 was modified correctly.
  - (l) Verifies that the line data for addr x+0x20 was unmodified.
  - (m) Verifies that the Central Cache tag for addr x+0x10 was invalidated.

Upon error, the test loops through steps a - m with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad system memory where block write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad system memory where no write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Central Cache tag not invalidated:
testaddr=0x00000000, tagaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that shown for the Block Copy (Source=Cache Miss, Dest=Cache Miss) Test.



### Block Copy (Source=Cache Hit, Dest=Cache Miss) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with a valid and clean Central Cache entry for the source causes the proper block of data to be transferred from Central Cache to memory and that the Central Cache block entry for the source is unchanged. The test does the following:

Turns off EN\_CACHE, EN\_IOPCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.

Clears Central Cache tags, data, and the first 64 Kbytes of system memory.

For each test address 0x0, 0x10, 0x20, 0x40, ..., 0x08000000 (128MB), or up to maximum memory installed, the test:

- (a) Writes the line for addr x (test addr) in memory with 00-0f data.
- (b) Writes the line for addr x+0x10 in memory with 50-5f data.
- (c) Writes the line for addr x+0x20 in memory with all zero data.
- (d) Writes the Central cache line for addr x with ff-f0 data.
- (e) Writes the Central cache line for addr x+0x10 with af-a0 data.
- (f) Writes the Central cache tag for addr x valid, clean.
- (g) Turns on Central Cache.
- (h) Does a block copy read from addr x.
- (i) Does a block copy write to addr x+0x10.
- (j) Turns off Central Cache.
- (k) Verifies that the line data for addr x was unmodified.
- (l) Verifies that the line data for addr x+0x10 was modified correctly.
- (m) Verifies that the line data for addr x+0x20 was unmodified.
- (n) Verifies that the Central Cache tag for line addr x was unchanged.

Upon error, the test loops through steps a - n with a constant test address.  
Possible error messages for this test are:

error1: Bad system memory where no write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad system memory where block write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad system memory where no write expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error4: Central Cache tag UNEXPECTEDLY invalidated:  
testaddr=0x00000000, tagaddr=0x00000000, exp=0x00000000, obs=0x00000000

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 14          | 0x0e                      | bit 7 ○○○○●●○ bit 0   | okay      |
| 14          | 0x8e                      | ●○○○●●○               | error     |

### Block Copy (Source=Cache Hit, Dest=Cache Hit) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with the a valid and clean Central Cache entry for the source and destination causes the proper block of data to be transferred from Central Cache to memory and that the Central Cache block entry for the destination address is invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0, 0x10, 0x20, 0x40, ..., 0x08000000 (128MB), or up to maximum memory installed, the test:
  - (a) Writes the line for addr x (test addr) in memory with 00-0f data.
  - (b) Writes the line for addr x+0x10 in memory with 50-5f data.
  - (c) Writes the line for addr x+0x20 in memory with all zero data.
  - (d) Writes the Central cache line for addr x with ff-f0 data.
  - (e) Writes the Central cache line for addr x+0x10 with af-a0 data.
  - (f) Writes the Central cache tag for addr x, x+0x10 valid, clean.
  - (g) Turns on Central Cache.
  - (h) Does a block copy read from addr x.
  - (i) Does a block copy write to addr x+0x10.
  - (j) Turns off Central Cache.
  - (k) Verifies that the line data for addr x was unmodified.
  - (l) Verifies that the line data for addr x+0x10 was modified correctly.
  - (m) Verifies that the line data for addr x+0x20 was unmodified.
  - (n) Verifies that the Central Cache tag for addr x is unchanged.
  - (o) Verifies that the Central Cache tag for addr x+0x10 was invalidated.

Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad system memory where block write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad system memory where no write expected:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Central Cache tag UNEXPECTEDLY invalidated:
testaddr=0x00000000, tagaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error5: Central Cache tag not invalidated:
testaddr=0x00000000, tagaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display for this test is the same as that for the Block Copy (Source=Cache Hit, Dest=Cache Miss) Test.



### Memory Write/Write/Read Read Test (Central Cache on)

This test verifies the address and data paths to Main Memory, as well as address bit and data bit uniqueness.

For each test address of main memory:  
(0x0,0x1,0x2,0x4,0x8,0x10,0x20....size\_of\_mem\_installed)

For each data pattern at each test address:  
(0x0,0x1,0x2,0x4,0x10,0x20....0x80000000)

The test does the following:

- (a) Writes test data to test address.
- (b) Writes inverted test data to test address + 0x04.
- (c) Reads back data from test address and compares.

### CAUTION THIS RAM TEST IS BEING DONE WITH CENTRAL CACHE ENABLED!

Upon error: loop through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 15          | 0x0f                      | bit 7 ○○○○●●● bit 0   | okay      |
| 15          | 0x8f                      | ●○○○●●●               | error     |

### IOC Tag RAM Write/Write/Read Test

This test performs three passes of write/write/read testing on the entire IOC Tag RAM. Each pass consists of writing a longword test pattern to the address under test, then the 1's complement of that pattern to the next long word, and finally, reading the original long word back and comparing it with what was written. The test address is then incremented by one longword and the process is repeated until the end of RAM is reached. The first pass is done with a test pattern of 0x5a972c5a, the second pass is done with a test pattern of 0x5a5a972c, and the third pass is done with a test pattern of 0x2c5a5a97.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 16          | 0x10                      | bit 7 ○○○●○○○ bit 0   | okay      |
| 16          | 0x90                      | ●○○●○○○               | error     |

**IOC Tag RAM Address Test**

This test writes the complete I/O Cache Tag RAM address space with the long-word address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Cache Tag RAM.

This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display is the same as that shown for the IOC Tag RAM Write/Write/Read Test.

**IOC Tag RAM 3-Pattern Test**

This test writes the complete I/O Cache Tag RAM address space with a repeated three long word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display is the same as that shown for the IOC Tag RAM Write/Write/Read Test.

**IOC Data RAM  
Write/Write/Read Test**

This test performs 3 passes of write/write/read testing on the entire IOC Data RAM. Each pass consists of writing a longword test pattern to the address under test, then the 1's complement of that pattern to the next long word, and finally, reading the original long word back and comparing it with what was written. The test address is then incremented by one longword and the process is repeated until the end of RAM is reached. The first pass is done with a test pattern of 0x5a972c5a, the second pass is done with a test pattern of 0x5a5a972c, and the third pass is done with a test pattern of 0x2c5a5a97.

This test enters a scope loop on data compare errors, with the following terminal message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 17                 | 0x11                             | bit 7 ○○○●○○○ bit 0          | okay             |
| 17                 | 0x91                             | ●○○●○○○                      | error            |



**IOC Data RAM Address Test**

This test writes the complete I/O Cache Data RAM address space with the long-word address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Cache Data RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

**IOC Data RAM 3-Pattern Test**

This test writes the complete I/O Cache Data RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

**IOC Data RAM Read Byte Alignment Test**

This test verifies that byte, word, and long-word read operations produce the appropriate byte-aligned data. The test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

**IOC Data RAM Write Byte Alignment Test**

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations. The test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

## VME Loopback Test

This test verifies that the VME loopback function works for writes and reads. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. For each test address (0x0, 0x4, 0x8, 0x10, 0x20, 0x00040000....0x00080000) and for each data pattern (0x0, 0x1, 0x2, ..., 0x80000000) at each test address, the test:
  - (a) Turns on VME-loopback in System Enable Register.
  - (b) Writes the address, using DMVA offset.
  - (c) Reads the address, using DVMA offset.
  - (d) Reads the address again, using DVMA offset.
  - (e) Turns off VME-loopback in System Enable Register.
  - (f) Verifies that the data read is the same as the data that was written.

Upon error: loop through steps a - f with a constant address and data. Possible error messages for this test are:

```
error1: testaddr=0x00000000, testdata=0x00000000, exp=0x00000000, obs=0x00000000
```

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 18                 | 0x12                             | bit 7 ○○○●○○○ bit 0          | okay             |
| 18                 | 0x92                             | ●○○●○○○                      | error            |



## VME Loopback and DVMA Test

This test is not used on a Sun-3/460 workstation.

This test verifies that the VME loopback function works for DVMA writes and reads.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOPCACHE, EN\_DVMA, EN\_VME\_LOOP in System Enable Register.
2. For each test address (0x0, 0x4, 0x8, 0x10, 0x20, 0x00040000....0x00080000) and for each data pattern (0x0, 0x1, 0x2....0x80000000) at each test address, the test does the following:
  - (a) Writes IO Mapper entry for test address.
  - (b) Turns on DVMA, VME-loopback in System Enable Register.
  - (c) Writes the address, using DVMA offset.
  - (d) Reads the address, using DVMA offset.
  - (e) Turns off DVMA in System Enable Register.
  - (f) Reads the address again, using DVMA offset.
  - (g) Turns off VME-loopback in System Enable Register.
  - (h) Verifies that the data read is the same as the data that was written.

Upon error, the test loops through steps a - h with constant addresses and data. Possible error messages for this test are:

error1: testaddr=0x00000000, testdata=0x00000000, exp=0x00000000, obs=0x00000000

The diagram below summarizes the LED states.

| <i>Test Number</i> | <i>Hexadecimal Value Of LEDs</i> | <i>Visual Representation</i> | <i>Condition</i> |
|--------------------|----------------------------------|------------------------------|------------------|
| 19                 | 0x13                             | bit 7 ●●●●●●● bit 0          | okay             |
| 19                 | 0x93                             | ●●●●●●●                      | error            |

## IOC Read Hit Test

This test verifies that a data operand read from system memory with a valid memory block address in the IO Cache causes a read from the IO Cache and not from system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and 1st 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10....0x1000, 0x2000, 0x2004....0x3000....0x000f1000). the test does the following:
  - (a) Write the longword address as data into memory.
  - (b) Write the inverted longword address as data into cache data ram.
  - (c) Write the address, valid bit ON in cache tag ram.
  - (d) Write IO Mapper entry for test address.
  - (e) Turn on IO Cache, DVMA, VME-loopback in System Enable Register.
  - (f) Read the address, using DVMA offset.
  - (g) Turn off IO Cache, DVMA in System Enable Register.
  - (h) Read the address again, using DVMA offset.
  - (i) Turn off VME-loopback in System Enable Register.
  - (j) Verify that the data read is the inverse of the address, and therefore, is not read from system memory, but rather from cache.
  - (k) Verify cache tag is still correct (valid, unmodified).
  - (l) Verify cache data is still correct (unmodified).

Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

error1: Bad read data on cache valid read:  
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad IOC tag ram after cache valid read:  
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad IOC data ram after cache valid read:  
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The diagram below summarizes the LED states.

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 20          | 0x14                      | bit 7 ●○○●○○● bit 0   | okay      |
| 20          | 0x94                      | ●○○●○○●               | error     |



## IOC Invalid Read Miss Test

This test verifies that a data operand read from system memory with an invalid memory block address in the IO Cache causes a read from system memory and not from the IO Cache. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and 1st 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10....0x1000, 0x2000, 0x2004....0x3000....0x000f1000), the test does the following:
  - (a) Writes 00-0f as data into memory for cache line of interest.
  - (b) Writes ff-f0 as data into cache data ram for cache line of interest.
  - (c) Writes the address, valid bit OFF in cache tag ram.
  - (d) Writes IO Mapper entry for test address.
  - (e) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
  - (f) Reads the address, using DVMA offset.
  - (g) Turns off IO Cache, DVMA in System Enable Register.
  - (h) Reads the address again, using DVMA offset.
  - (i) Turns off VME-loopback in System Enable Register.
  - (j) Verifies that the data read is the same as the address, and therefore is not read from io cache, but rather from system memory.
  - (k) Verifies cache tag is updated correctly.
  - (l) Verifies cache data is updated correctly (entire line).

Upon error: loop through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after cache invalid read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC data ram after cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

## IOC Write Hit Test

This test verifies that a data operand write to system memory with a valid memory block address in the IO Cache causes a write to the IO Cache and not to system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000). the test does the following:
  - (a) Write 00-0f (16 bytes) into memory for cache line of interest.
  - (b) Write all zeros into cache data ram for cache line of interest.
  - (c) Write the address, valid bit ON in cache tag ram.
  - (d) Write IO Mapper entry for test address.
  - (e) Turn on IO Cache, DVMA, VME-loopback in System Enable Register.
  - (f) Write address (with DVMA offset) with inverse address as data.
  - (g) Turn off IO Cache, DVMA, VME-loopback in System Enable Register.
  - (h) Verify that system memory location was unmodified, since the write should have gone only to the cache (no write-through).
  - (i) Verify cache tag was updated correctly (valid, dirty, addr bits).
  - (j) Verify cache data was written correctly.

Upon error: loop through steps a - j with a constant test address. Possible error messages for this test are:

error1: Bad system memory after cache valid write hit:

testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad IOC tag ram after cache valid write hit:

testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad IOC data ram after cache valid write hit:

testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The LED display is the same as that for the IOC Read Hit Test.



## IOC Write Miss, No Writeback Test

This test verifies that a data operand write to system memory with an invalid memory block address in the IO cache causes a write to IO cache and not to system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes 00-0f (16 bytes) into memory for cache line of interest.
  - (b) Writes all zeros into cache data ram for cache line of interest.
  - (c) Writes the address, valid bit OFF in cache tag ram.
  - (d) Writes IO Mapper entry for test address.
  - (e) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
  - (f) Writes address (with DVMA offset) with inverse address as data.
  - (g) Turns off IO Cache, DVMA, VME-loopback in System Enable Register.
  - (h) Verifies that system memory location was unmodified, since the write should have gone only to the cache (no write-through).
  - (i) Verifies cache tag was updated correctly (valid, dirty, addr bits).
  - (j) Verifies cache data was written correctly.

Upon error: loop through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache write miss:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad IOC tag ram after cache write miss:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad IOC data ram after cache write miss:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

**IOC Write Miss, Writeback Test**

This test verifies that a data operand write to system memory with a different, valid and dirty memory block address in the IO cache causes a writeback to system memory.

The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears the Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000) the test:
  - (a) Writes the line for memory address x (writeback addr) with ff-f0 data.
  - (b) Writes the line for memory address x+0x10 with zero data.
  - (c) Writes the line in data cache for addr x with 00-0f data.
  - (d) Writes the address (x), VALID, DIRTY in cache tag RAM.
  - (e) Writes an IO Mapper entry for the test address.
  - (f) Turns on IO Cache, DVMA, VME-loopback in the System Enable Reg.
  - (g) Writes the address (with DVMA offset) with inverse address as data.
  - (h) Turns off the IO Cache, DVMA, VME-loopback in System Enable Reg.
  - (i) Verifies that the line for addr x was written back to memory.
  - (j) Verifies that system memory for location (x+0x10) was unmodified.
  - (k) Verifies that the cache tag was updated correctly (valid, dirty, addr bits).
  - (l) Verifies that cache data was written correctly.

Upon error, the test loops through steps a - l with a constant test address.

Possible error messages for this test are:

```
error1: Bad system memory where writeback should have occurred:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC tag ram after cache write miss:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Bad IOC data ram after cache write miss:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



## IOC Read Miss, Writeback Test

This test Verifies that a data operand read from system memory with a different valid and dirty memory block address in the IO cache causes a writeback to system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000).

the test does the following:

- (a) Writes the line for memory address x (writeback addr) with ff-f0 data.
- (b) Write the line for memory address x+0x10 with 50-5f data.
- (c) Writes the line in data cache for addr x with 00-0f data.
- (d) Writes the address (x), VALID, DIRTY in cache tag ram.
- (e) Writes the IO Mapper entry for the test address.
- (f) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
- (g) Reads from address x+0x10 (with DVMA offset).
- (h) Turns off IO Cache, DVMA in System Enable Register.
- (i) Reads from address x+0x10 (with DVMA offset).
- (j) Turns off VME-loopback in System Enable Register.
- (k) Verifies data read in was from memory, not cache.
- (l) Verifies that the line for addr x was written back to memory.
- (m) Verifies that the system memory line for addr (x+0x10) was unmodified.
- (n) Verifies that the cache tag was updated correctly (valid,addr bits).
- (o) Verifies that cache data was written correctly.

Upon error, the test loops through steps a - o with a constant test address.  
Possible error messages for this test are:

```
error1: Bad read data on cache valid read miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error2: Bad system memory where writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error3: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error4: Bad IOC tag ram after cache valid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

```
error5: Bad IOC data ram after cache valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

### IOC Valid Write Hit (Central Cache Match, Unmod) Test

This test Verifies that a data operand write to system memory with a valid memory block address in the IO Cache *and* in the Central Cache causes a write to the IO Cache and not to system memory, and that the entry in the Central Cache is invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000).

the test does the following:

- (a) Writes the line for memory address x with 00-0f data.
- (b) Writes the line in IO cache for addr x with all zero data.
- (c) Writes the address (x), VALID in IO cache tag ram.
- (d) Writes the address (x), VALID in Central cache tag ram.
- (e) Writes the IO Mapper entry for the test address.
- (f) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
- (g) Writes to the test address (with DVMA offset) with the inverted address as data.
- (h) Turns off the Central Cache, IO Cache, DVMA, and VME-loopback in the System Enable register.
- (i) Verifies that the system memory line for addr x was unmodified.
- (j) Verifies that the IO cache tag was updated correctly (valid, dirty, address bits).
- (k) Verifies that the IO cache data was written correctly.
- (l) Verifies that the Central Cache tag was updated correctly (invalid).

Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after cache valid write hit:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC data ram after cache valid write hit:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Bad Central Cache tag ram after cache valid write hit:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



### IOC Invalid Write Miss (Central Cache Match, Unmod) Test

This test verifies that a data operand write to system memory with a memory block address that is INVALID in the IO Cache and VALID in the Central Cache causes a write to IO Cache and not to system memory, and that the entry in the Central Cache is invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000).

the test does the following:

- (a) Writes the system memory line for addr x with 00-0f data.
- (b) Writes the line in IO cache for addr x with all zero data.
- (c) Writes the address (x), INVALID in IO cache tag ram.
- (d) Writes the address (x), VALID in Central cache tag ram.
- (e) Writes the IO Mapper entry for the test address.
- (f) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
- (g) Writes to the test address (with DVMA offset) with the inverted address as data.
- (h) Turns off Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
- (i) Verifies that the system memory line for addr x was unmodified.
- (j) Verifies that the IO cache tag was updated correctly (valid, dirty, address bits).
- (k) Verifies that IO cache data was written correctly.
- (l) Verifies that Central Cache tag was updated correctly (invalid).

Upon error, the test loops through steps a - l with a constant test address.

Possible error messages for this test are:

error1: Bad system memory after cache invalid write miss:

testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad IOC tag ram after cache invalid write miss:

testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad IOC data ram after cache invalid write miss:

testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error4: Bad Central Cache tag ram after cache invalid write miss:

testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The LED display is the same as that for the IOC Read Hit Test.

# IOC Invalid Read Miss (Central Cache Match, Unmod) Test

This test verifies that a data operand read from system memory with the memory block address that is INVALID in the IO Cache and VALID in the Central Cache causes a read from Central Cache and not from IO cache or system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000).

the test does the following:

- (a) Writes the system memory line for addr x with ff-f0 data.
- (b) Writes the line in IO cache for addr x with all zero data.
- (c) Writes the address (x), INVALID in IO cache tag ram.
- (d) Writes the line in Central cache for addr x with 00-0f data.
- (e) Writes the address (x), VALID in Central cache tag ram.
- (f) Writes the IO Mapper entry for test address.
- (g) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
- (h) Reads from the test address (with DVMA offset).
- (i) Turns off Central Cache, IO Cache, DVMA in the System Enable register.
- (j) Reads from the test address again (with DVMA offset).
- (k) Turns off VME-loopback in the System Enable register.
- (l) Verifies that the data read in is correct (came from Central cache).
- (m) Verifies that the system memory line for addr x was unmodified.
- (n) Verifies that the IO cache tag was updated correctly (valid, clean, address bits).
- (o) Verifies that the IO cache data was written correctly.
- (p) Verifies that the Central cache tag was unchanged.



Upon error, the test loops through steps a - p with a constant test address.  
Possible error messages for this test are:

error1: Bad read data on cache invalid read:

testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory after cache invalid read miss:

testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC tag ram after cache invalid read miss:

testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC data ram after cache valid read miss:

testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Bad Central Cache tag ram after cache invalid read miss:

testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

The LED display is the same as that for the IOC Read Hit Test.

# IOC Invalid Read Miss (Central Cache Match,Modified) Test

This test verifies that a data operand read from system memory with a memory block address that is INVALID in the IO Cache; VALID in the Central Cache; and MODIFIED causes a read from Central Cache and not from IO cache or system memory. The test does the following:

1. Turns off EN\_CACHE,EN\_IOCACHE,EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:
  - (a) Writes the system memory line for addr x with ff-f0 data.
  - (b) Writes the line in IO cache for addr x with all zero data.
  - (c) Writes the address (x), INVALID in IO cache tag ram.
  - (d) Writes the line in Central cache for addr x with 00-0f data.
  - (e) Writes the address (x), VALID, DIRTY in Central cache tag ram.
  - (f) Writes the IO Mapper entry for the test address.
  - (g) Turns on Central Cache, IO Cache, DVMA and VME-loopback in the System Enable register.
  - (h) Reads from the test address (with DVMA offset).
  - (i) Turns off Central Cache, IO Cache, DVMA in the System Enable register.
  - (j) Reads from the test address (with DVMA offset).
  - (k) Turns off VME-loopback in System Enable register.
  - (l) Verifies that the data read in is correct (came from Central cache).
  - (m) Verifies that the system memory line for addr x was unmodified.
  - (n) Verifies that the IO cache tag was updated correctly (valid, clean, address bits).
  - (o) Verifies that IO cache data was written correctly.
  - (p) Verifies that the Central cache tag was unchanged.
  - (q) Verifies that the Central cache data was unchanged.

Upon error, the test loops through steps a - q with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad system memory after cache invalid read miss:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC tag ram after cache invalid read miss:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Bad IOC data ram after cache valid read miss:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error5: Bad Central Cache tag ram after cache invalid read miss:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



# IOC Valid Read Miss (Central Cache Match), Writeback Test

This test verifies that a data operand read from system memory with a memory block address that is VALID,MODIFIED in the IO Cache and another that is VALID in the Central Cache causes a read from Central Cache and not from IO cache or system memory. Also, a writeback of the valid and modified data in the IOC should take place. The test does the following:

1. Turns off EN\_CACHE,EN\_IOCACHE,EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:
  - (a) Writes the system memory line for addr x with ff-f0 data.
  - (b) Writes the system memory line for addr x+0x10 with all zero data.
  - (c) Writes the line in IO cache for addr x with 50-5f data.
  - (d) Writes the line in Central cache for addr x with 00-0f data.
  - (e) Writes the address (x+0x10), VALID,DIRTY in IO Cache tag RAM.
  - (f) Writes the address (x), VALID in Central Cache tag RAM.
  - (g) Writes the IO Mapper entry for test address.
  - (h) Turns on Central Cache, IO Cache, DVMA and VME-loopback in the System Enable register.
  - (i) Reads from test address x (with DVMA offset).
  - (j) Turns off Central Cache, IO Cache and DVMA in the System Enable register.
  - (k) Reads from test address x (with DVMA offset).
  - (l) Turns off VME-loopback in the System Enable register.
  - (m) Verifies that data read in is correct (came from Central cache).
  - (n) Verifies that the system memory line for addr x was unmodified.
  - (o) Verifies that a writeback took place to system memory address x+0x10.
  - (p) Verifies that the IO cache tag was updated correctly (valid,clean,address bits).
  - (q) Verifies that the IO cache data was written correctly.
  - (r) Verifies that the Central cache tag was unchanged.
  - (s) Verifies that the Central cache data was unchanged.

Upon error, the test loop through steps a - s with a constant test address.  
Possible error messages for this test are:

error1: Bad read data on IOC valid read miss:  
testaddr=0x00000000, readaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad system memory where NO writeback expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad system memory where WRITEBACK expected:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error4: Bad IOC tag ram after IOC valid read miss:  
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error5: Bad IOC data ram after IOC valid read miss:  
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error6: Bad Central Cache tag ram after IOC valid read miss:  
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error7: Bad Central Cache data ram after IOC valid read miss:  
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The LED display is the same as that for the IOC Read Hit Test.



## IOC Flush (Valid, Modified) Test

This test verifies that a flush cache block write with valid and dirty data in the IO cache causes the line to be written back to memory and the entry in the IO cache to be invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes the line for memory address x with 00-0f data.
  - (b) Writes the line in data cache for addr x with ff-f0 data.
  - (c) Writes the address (x), VALID, DIRTY in cache tag RAM.
  - (d) Writes IO Mapper entry for test address.
  - (e) Turns on IO Cache, DVMA, VME-loopback in the System Enable register.
  - (f) Writes to the flush block addr for the test address.
  - (g) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.
  - (h) Verifies that the line for addr x was written back to memory.
  - (i) Verifies that the IOC tag was updated correctly (invalid).
  - (j) Verifies that the IOC data RAM remained unchanged.

Upon error, the test loops through steps a - j with a constant test address.  
Possible error messages for this test are:

error1: Bad system memory where writeback should have occurred:  
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000

error2: Bad IOC tag ram after IOC block flush:  
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000

error3: Bad IOC data ram after IOC block flush:  
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000

The LED display is the same as that for the IOC Read Hit Test.

## IOC Flush (Valid, Not Modified) Test

This test verifies that a flush cache block write with valid and clean data in the IO cache causes the line NOT to be written back to memory and the IO cache entry to be invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes the line for memory address x with 00-0f data.
  - (b) Writes the line in data cache for addr x with ff-f0 data.
  - (c) Writes the address (x), VALID, in cache tag ram.
  - (d) Writes the IO Mapper entry for the test address.
  - (e) Turns on the IO Cache, DVMA, VME-loopback in the System Enable register.
  - (f) Writes to the flush block address for the test address.
  - (g) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
  - (h) Verifies that the line for address x was NOT written back to memory.
  - (i) Verifies that the cache tag was updated correctly (invalid).
  - (j) Verifies that IOC data RAM remained unchanged.

Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after IOC block flush:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC data ram after IOC block flush:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



## IOC Flush (Not Valid, Not Modified) Test

This test verifies that a flush cache block write with invalid data in the IO cache causes the NOT to be written back to memory and the entry in the IO cache to remain invalidated. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes the line for memory address x with 00-0f data.
  - (b) Writes the line in data cache for addr x with ff-f0 data.
  - (c) Writes the address (x), INVALID, in cache tag ram.
  - (d) Writes the IO Mapper entry for the test address.
  - (e) Turns on the IO Cache, DVMA and VME-loopback in the System Enable register.
  - (f) Writes to the flush block address for the test address.
  - (g) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
  - (h) Verifies that the line for addr x was NOT written back to memory.
  - (i) Verifies that the cache tag remained invalid.
  - (j) Verifies that IOC data RAM remained unchanged.

Upon error, the test loops through steps a - j with a constant test address.  
Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after IOC block flush:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC data ram after IOC block flush:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

## I/O Mapper Invalid Page (IO.DT) Test

This test verifies that a data operand read from system memory with the memory block address in the IO Cache INVALID and the IO mapper entry for that page set to INVALID causes an access violation, and that there is no data cached from system memory. For each of (IO.DT = 00, 10, 11), the test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes ff-f0 as data into memory for cache line of interest.
  - (b) Writes 00-0f as data into cache data RAM for cache line of interest.
  - (c) Writes the address, valid bit OFF in cache tag ram.
  - (d) Writes the IO Mapper entry for the test address; sets INVALID page.
  - (e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
  - (f) Reads the address, using DVMA offset.
  - (g) Turns off IO Cache and DVMA in the System Enable register.
  - (h) Reads the address, using DMVA offset.
  - (i) Turns off the VME-loopback in System Enable register.
  - (j) Verifies that the cache tag is unchanged.
  - (k) Verifies that the cache data is unchanged.

Upon error, the test loops through steps a - k with a constant test address.

Possible error messages for this test are:

```
error1: Bad IOC tag ram after cache invalid (page invalid) read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC data ram after cache invalid (page invalid) read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



### IOC Write Miss, Writeback (Write Protect) Test

This test verifies that a data operand write to system memory with a different memory block address in the IO cache valid and dirty and the IO mapper entry for the block's page set to Write Protect (IO.WP = 1) causes NO writeback to system memory. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Turns off IO cache.
  - (b) Writes the line for memory address x (writeback addr) with ff-f0 data.
  - (c) Writes the line for memory address x+0x10 with 50-5f data.
  - (d) Writes the line in data cache for addr x with 00-0f data.
  - (e) Writes the address (x), VALID, DIRTY in cache tag RAM.
  - (f) Writes the IO Mapper entry for the test address.
  - (g) Turns on the IO Cache, DVMA and VME-loopback in the System Enable register.
  - (h) Writes the address (with DVMA offset) with inverse address as data.
  - (i) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.
  - (j) Verifies that the line for addr x was NOT written back to memory.
  - (k) Verifies that system memory for location (x+0x10) was unmodified.
  - (l) Verifies that the cache tag was NOT updated.
  - (m) Verifies that cache data was NOT updated.

Upon error, the test loops through steps b - m with a constant test address.  
Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should occur (write protect):
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad system memory where no write should have occurred (write protect):
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error3: Bad IOC tag ram after cache write miss (write protect):
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error4: Bad IOC data ram after cache write miss (write protect):
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

# IOC Invalid Read Miss (IO Mapper IO.EN = 0) Test

This test verifies that a data operand read from system memory with an invalid memory block address in the IO Cache *and* the IO.EN bit in the IO Mapper entry for this page set OFF causes a read from system memory and not from the IO Cache, but that nothing gets modified in the IOC data and tags. The test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Writes 00-0f as data into memory for cache line of interest.
  - (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
  - (c) Writes the address, valid bit OFF in cache tag RAM.
  - (d) Writes the IO Mapper entry for the test address, IO.EN = 0.
  - (e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
  - (f) Reads the address, using DVMA offset.
  - (g) Turns off IO Cache and DVMA in the System Enable register.
  - (h) Reads the address again, using DMVA offset.
  - (i) Turns off VME-loopback in the System Enable register.
  - (j) Verifies that the data read is the same as the address, and therefore is not read from IO Cache, but rather from system memory.
  - (k) Verifies that the IOC tag is unchanged (still invalid).

Upon error, the test loops through steps a - k with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000, dataramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after cache invalid read:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



# IOC Write Miss (IO Mapper IO.EN = 0) Test

This test verifies that a data operand write to system memory with an invalid memory block address in the IO cache *and* the IO.EN bit in the IO Mapper entry for this page set OFF causes a write to system memory and not to the IOC. This test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0, 0x4, 0x8, 0x10, ..., 0x1000, 0x2000, 0x2004, ..., 0x3000, ..., 0x000f1000), the test does the following:
  - (a) Write all zeroes into memory line for test address.
  - (b) Write 00-0f data into IOC data ram for cache line of interest.
  - (c) Write the address, valid bit OFF in cache tag ram.
  - (d) Write IO Mapper entry for test address, IO.EN = 0.
  - (e) Turn on IO Cache, DVMA, VME-loopback in System Enable reg.
  - (f) Write address (with DVMA offset) with inverse address as data.
  - (g) Turn off IO Cache, DVMA, VME-loopback in System Enable reg.
  - (h) Verifies that system memory location was modified, since the write should have gone to memory (cache disabled for this page).
  - (i) Verifies that the cache tag was not modified (still invalid).

Upon error, the test loops through steps a - i with a constant test address.  
Possible error messages for this test are:

```
error1: Bad system memory after cache write miss:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

```
error2: Bad IOC tag ram after cache write miss:
testaddr=0x00000000, tagramaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

## IOC Random Data Block Write Test

This test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0,0x4,0x8,0x10,0x20,0x40,0x80,...0x100000, the test:
  - (a) Clears out all IOC tags, data.
  - (b) Clears out the entire IO Mapper.
  - (c) Writes eight IO Mapper entries for the 64MB space to be tested (x-->x+64ME with address, IO.DT, IO.EN.
  - (d) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
  - (e) Copies the entire 64KB EPROM contents to memory starting at the test address, doing DVMA writes through the IOC, flushing the last line of each 8K block.
  - (f) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
  - (g) Verifies that 64KB of system memory matches EPROM contents.

Upon error, the test loops through step (g) with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after block write:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



IOC Random Data Block Read  
(Central Cache off) Test

This test does the following:

1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, ... 0x100000, the test:
  - (a) Clears out all IOC tags, data.
  - (b) Clears out entire IO Mapper.
  - (c) Writes eight IO Mapper entries for 64MB space to be tested (x-->x+64MB), with address, IO.DT, IO.EN.
  - (d) Copies the entire 64KB EPROM contents to memory, starting at the test address.
  - (e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
  - (f) Reads 64KB of data back from memory by way of DVMA; reads one longword at a time and compares with contents of EPROM.
  - (g) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.

Upon error, the test loops through step (f) with a constant test address. Possible error messages for this test are:

```
error1: Bad IOC read data:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

### IOC Random Data Block Read (Central Cache on) Test

This test does the following: IP 1. Turns off EN\_CACHE, EN\_IOCACHE, EN\_DVMA, and EN\_VME\_LOOP in the System Enable Register.

2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, ... 0x100000, the test:
  - (a) Clears out all IOC tags, data.
  - (b) Clears out entire IO Mapper.
  - (c) Writes eight IO Mapper entries for 64MB space to be tested (x-->x+64MB), with address, IO.DT, IO.EN.
  - (d) Copies the entire 64KB EPROM contents to memory, starting at test address.
  - (e) Turns on IO Cache, DVMA, VME-loopback, Central Cache in SYSENREG.
  - (f) Reads 64KB of data back from memory, using DVMA, reading one longword at a time and comparing with EPROM contents.
  - (g) Turns off IO Cache, DVMA, VME-loopback and Central Cache in the System Enable register.

Upon error, the test loops through step (f) with a constant test address. Possible error messages for this test are:

```
error1: Bad IOC read data:
testaddr=0x00000000, memaddr=0x00000000, exp=0x00000000, obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.



# P4 Overlay Frame Buffer Write/Write/Read Test

This test will be executed only if a P4 Video RAM board is detected. First, the test probes for a P4 board, with the following possible messages:

<P4 High Resolution Monochrome Video RAM Board Detected>

<P4 Low Resolution Monochrome Video RAM Board Detected>

<P4 Low Res Color Video RAM Board Detected>

<ILLEGAL P4 Video RAM Board ID Detected: 0x00000000>

<P4 Video RAM Board NOT Detected>

The same test is executed if either the monochrome or color board is detected. The test verifies the address and data paths to the P4 Overlay Frame Buffer, as well as address bit and data bit uniqueness.

For each test address of P4 Overlay Frame Buffer, the test does the following, for each data pattern at each test address

(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000):

(a) Writes test data to test address.

(b) Writes inverted test data to test address + 0x04.

(c) Reads back data from test address and compares.

Upon error, the test loop through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED indications are as follows:

| Test Number | Hexadecimal Value Of LEDs | Visual Representation | Condition |
|-------------|---------------------------|-----------------------|-----------|
| 21          | 0x15                      | blt 7 ○○○○●○○● blt 0  | okay      |
| 21          | 0x95                      | ●○○○●○○●              | error     |

#### P4 Overlay Frame Buffer Address Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_ADDR_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected.

This test writes the complete P4 Overlay Frame Buffer address space with the longword address as data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the P4 Overlay Frame Buffer RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

#### P4 Overlay Frame Buffer 3- Pattern Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_3PATT_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected. This test writes the complete P4 Overlay Frame Buffer address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.



#### P4 Overlay Frame Buffer March Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_MARCH_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected. A background of longword 0's is written to all of the P4 Overlay Frame Buffer RAM, starting from address 0 and ending at the highest address. Then the longword data is read at the first address and a 0xffffffff is written into this address. The same two-step read/write procedure is continued at each sequential longword until the end of memory is reached. Then each longword is tested and changed back to zero in reverse order until the first address is reached. Finally, the same sequence is repeated using complemented data (i.e. a background of 1's is written into memory).

Upon error, the test loops on a complete write then read of Video RAM. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

#### P4 Overlay Frame Buffer Read Byte Alignment Test

The same test is executed if either the monochrome or color board is detected. This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

This test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

#### P4 Overlay Frame Buffer Write Byte Alignment Test

The same test is executed if either the monochrome or color board is detected. This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

## P4 Enable Plane Write/Write/Read Test

This test will be executed only if a P4 Low Res Color Video RAM board is detected. The test verifies the address and data paths to the P4 Enable Plane RAM, as well as address bit and data bit uniqueness. For each test address of P4 Enable Plane

(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+ram\_size), the test does the following:

For each data pattern at each test address:

(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)

- (a) Writes test data to the test address.
- (b) Writes the inverted test data to the test address + 0x04.
- (c) Reads the data back from the test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.



P4 Color Plane  
Write/Write/Read Test

This test will be executed only if a P4 Low Res Color Video RAM board is detected. The test verifies the address and data paths to the P4 Color Plane RAM, as well as address bit and data bit uniqueness. For each test address of P4 Color Plane  
(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+ram\_size)

the test does the following:

For each data pattern at each test address:  
0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000), the test

- (a) Writes test data to the test address.
- (b) Write inverted test data to the test address + 0x04.
- (c) Reads the data back from the test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

Printer Controller Check —  
Sun-3/80

The *Printer Controller Check* tests the ability of the processor to access the Parallel Printer Controller registers.

An incrementing pattern is displayed on the data output bits at the parallel printer connector. Each pattern is read back through the internal registers and verified.

If an error is detected, the test will enter a scope-loop, writing and reading the failed pattern. Only the first error will be reported to the serial port.

Printer Controller Data Error Expected xxxxxxxx, OBS xxxxxxxx

Clock/Calendar Device — 3/80  
Only

This test verifies that the Clock/Calendar device is operational. The test is in several phases:

1. It verifies that the battery is not low:  
WARNING: The Clock/Calendar Battery is low.
2. It verifies that the seconds counter is updating (the clock is running). If the seconds counter is not updating, the device is reinitialized and its date and time set to January 1, 1989 00:00:00. Two warning messages may be displayed that indicate an incorrectly initialized device or possibly a failed device.

<Clock is not running!, re-starting TOD Clock Module.>

<New TOD Module installed, initializing and starting the clock

If an error is detected, the test will enter a scope-loop repeating the test sequence that failed. Only the first error will be reported to the serial terminal.

Time of Day Clock FAILED, Unit or support logic BAD

#### Configuration Memory Check — 3/80 Only

This test verifies the operation of the Configuration memory.

The Diagnostic Test locations in CMOS are tested and any errors detected are reported.

If an error is detected the test will enter a scope-loop, writing and reading the failed pattern. Only the first error will be reported to the serial port.

Data Path error Expected [xxxxxxx] Found [xxxxxxx]

Configuration RAM ERROR: Address uniqueness Fault

#### LANCE Ethernet Controller Check — 3/80 only

This test verifies the ability of the processor to access the controller's internal registers.

LANCE Controller BAD, unable to access.

#### ESP SCSI Check — 3/80 Only)

The ESP Controller Check includes the ability to access the ESP registers and the operation of the internal state-machine.

If an error is detected the test will enter a scope-loop repeating the test sequence that failed. Only the first error will be reported to the serial terminal.

ESP Controller Bad, FLAGS xxxxxxxx FIFO xxxxxxxx [xxxxxxx]

#### Host System Initialization

After the Sun-3 firmware has executed the power-up sequence, it initializes the host system. Devices initialized at this time are the TIA, TIB, Page Tables, PMMU, keyboard, mouse, serial ports, frame buffer, memory, and other CPU devices. Initialization tasks include memory sizing, interrupt vector setting, trap vector setting, and setting entry points that correspond to support routines.

#### The Sun-3 PROM Monitor

This section describes the PROM monitor (sometimes called the "system monitor") commands available on Sun-3 workstation. For those with Sun-3/400 series or Sun-3/80 workstation, it is intended to replace Chapter 8 of the *PROM User's Manual*.

Taken as a whole, the monitor commands offer a low-level user interface to the Sun hardware. They control a variety of options, including booting from an alternate device, changing the console output, reading or altering registers or memory locations, and so on.

The effects of most monitor commands disappear when the power is turned off, with the exception of the **q** command, which programs the EEPROM. Parameters entered with the **q** command remain until deliberately altered with another



PROM command. Programming the EEPROM *reconfigures* your workstation. The Boot PROM consults the EEPROM to determine whether or not to poll for a boot device, whether to boot a specified program, which device is the console, and so on. Refer to the **q** command in this chapter, for information on programming the EEPROM. Also refer to the `eeeprom` command described in the *SunOS Reference Manual*

## Bringing up the PROM Monitor

Read *Chapter 3* of the *PROM User's Manual* for instructions on bringing up the PROM monitor.

## Conventions

The paragraphs below describe each of the monitor commands in detail. Each paragraph starts with a line describing the command syntax. If a command has more than one distinct format, each one is shown on a separate line. Characters in **bold courier** font mean that you should enter them exactly as shown. Plain `courier` font represents what you should see on the screen or a program or path name. Words in *Roman italic* show the type of information you are to enter (variables), or list document names. *Roman italic* or **boldfaced** font is also used for notes or emphasis within the text. Optional arguments and default values are listed in the descriptions.

## Monitor Command Overview

The following example shows the menu that comes up when you enter **h** (help) at the monitor prompt. This section describes the general characteristics that all of the commands have in common. Detailed descriptions of each command are listed in the next section. Some commands are available only for Sun-3/400 series workstations, and those will be identified.



**NOTE** The *i*, *j*, *n*, and *y* commands are only available on systems with on-board cache memory, as indicated in *italics*.

Figure 6 Sun-3 Monitor Help Menu

| Boot PROM Monitor Commands                |                                                         |
|-------------------------------------------|---------------------------------------------------------|
| a [digit]                                 | Open CPU Addr Reg (0-7)                                 |
| b [dev([cntrl],[unit],[part])]            | Boot a file                                             |
| c [addr]                                  | Continue program at Addr                                |
| d [digit]                                 | Open CPU Data Reg (0-7)                                 |
| e [addr]                                  | Open Addr as 16 bit word                                |
| f beg_addr end_addr pattn [size]          | Fill Memory                                             |
| g [addr]                                  | Go to Addr                                              |
| h or ?                                    | Help Menu                                               |
| i [c] [i] [addr] <i>(Sun-3/470,3/260)</i> | Open Central or I/O Cache Data                          |
| j [c] [i] [addr] <i>(Sun-3/470,3/260)</i> | Open Central or I/O Cache Tags                          |
| k [number]                                | Reset (0)CPU, (1)MMU, (2)System                         |
| l [addr]                                  | Open Addr as 32 bit long                                |
| m [addr]                                  | Open Segment Map                                        |
| m [A] [B] [addr] <i>(Sun-3/470,3/80)</i>  | Display TIA or TIB Table Entries                        |
| n [i/e/d] <i>(not for Sun-3/470)</i>      | Cache Invalidate/Enable/Disable                         |
| o [addr]                                  | Open Addr as 8 bit byte                                 |
| p [addr]                                  | Open Page Map                                           |
| p [addr] <i>(for Sun-3/470)</i>           | Display Page Table (For Page Or I/O Map                 |
| q [addr]                                  | Open EEPROM                                             |
| r                                         | Open CPU/MMU Registers (i.e. PC)                        |
| s [digit]                                 | Set/Query Function Code (0-7)                           |
| t [y/n/c]                                 | Trace: Yes/No/Continue                                  |
| u [arg]                                   | Select Console Device                                   |
| v beg_addr end_addr [size]                | Display Memory                                          |
| w [addr] [string]                         | Vector                                                  |
| x                                         | Extended Diag Tests                                     |
| y [c cxt] [s cxt sg_addr] [p cxt pg_addr] | Flush Cntxt/Seg/Page <i>(not for Sun-3/470 or 3/80)</i> |
| z [addr]                                  | Set Breakpoint                                          |
| ~a <i>(Sun-3/80)</i>                      | Display physical/virtual address                        |
| ~f <i>(Sun-3/80)</i>                      | Flush ATC Cache                                         |
| ~r <i>(Sun-3/80)</i>                      | Read on-board device registers                          |

### Executing a Command

In general, to execute a command, you type the appropriate command letter, followed by any required command arguments. For example, if you wanted to execute the hypothetical command "0" (which we will say needs two arguments 100 and 200) you would type a line like this:

```
> 0 100 200
```

The command letter can be upper or lower case, and all commands and arguments are separated by white-space (tabs or spaces). Pressing the return key executes the command.



## Default Values

Many of the monitor commands have built in, or *default* values, which the command uses if you do not supply arguments. The default values vary from command to command. Check the command descriptions for the default values of interest.

## Word Sizes

Word sizes referred to in this chapter are defined as follows: A **byte** is eight bits long; a **word** is 16 bits long; a **long word** is 32 bits long.

## The Monitor Commands

This section provides more detailed information on the commands listed in the help menu example. Both the commands and their arguments are described here.

## Displaying and Modifying Memory

A number of the commands listed here may be used to display and/or modify the system's memory and registers. Regardless of the type of memory (RAM, EEPROM, etc.) or register, these commands have the same command syntax. This section describes the memory modification syntax used by the monitor commands. The example here references long words (32 bits) of memory, but the syntax is the same for bytes (8 bits) and words (16 bits).

The *address* argument specifies the initial memory location that is to be displayed and/or modified.

The second argument determines whether the current content of a memory location is to be displayed and/or modified. Entering *only* the address of a memory location after the command *displays* the content of that address.

```
>command FFE80000 
FFE80000: 12345678 ?
```

At this point, you can respond in one of *three* different ways.

1. Simply pressing the  key displays the contents of the next memory location (in this case, 0xFFE80004) as shown below.

```
>command FFE80000 
FFE80000: 12345678 ? 
FFE80004: 00000001 ?
```

Successively pressing the  key displays the contents of successive memory locations. Assuming that you pressed  four times, the contents of memory locations 0xFFE80004, 0xFFE80008, 0xFFE8000C and 0xFFE80010 would be displayed.



2. To exit the command, enter **any non-hexadecimal** character (q for quit, for example) before pressing **Return**. Note that you will now return to the monitor's basic command level, with the > prompt.

```
>command FFE80000 Return
FFE80000: 12345678 ? q Return
>
```

3. The *third* response to the display of memory location contents is to *modify* those contents. You enter the *new* hexadecimal value immediately following the question mark ?, BEFORE pressing **Return**. The following display demonstrates how the value of memory location 0xFFE80000 can be changed from "12345678" to "00ABCDEF".

**NOTE** Following the assignment of the new value to memory location 0xFFE80000, the new value will not be displayed; instead, the contents of the next memory location are shown. You will not be returned to the monitor's basic command level.

```
>command FFE80000 Return
FFE80000: 12345678 ? 00ABCDEF Return
FFE80004: 00000001 ?
```

Entering a memory location's virtual address followed *only* by a non-hexadecimal character before pressing the **Return** key causes the monitor to *display* the contents of that location then exit to the monitor command level.

```
>command FFE80004 q Return
FFE80004: 00000001
>
```

Entering a non-hexadecimal character *and* a hexadecimal value after an address causes the monitor to display the original value at that virtual address, assign a new value, then display that value. For instance, assume that the original content at address 0xFFE80010 is "00000005". The command

```
>command FFE80010 ? 55555550 Return
```

displays the original value "00000005" then assigns the value "55555550" to address FFE80010 before returning to the monitor prompt:

```
>command FFE80010 ? 55555550 Return
FFE80010: 00000005 -> 55555550
>
```



You may also enter multiple display and/or modify commands for multiple memory locations on the same command line. If you enter this command line:

```
>command FFE80000 ? 00000000 ? ? 22222220 33333330 q 
```

You will see this on the screen:

```
FFE80000: 12345678 -> 00000000
FFE80004: 00000001
FFE80008: 00000002 -> 22222220
FFE8000C -> 33333330
FFE80010: 00000004
>
```

The first part of the command line,

```
>command FFE80000 ? 00000000
```

displays the original contents of location FFE80000 before assigning the new value "00000000" to it.

The next question mark directs the monitor to display the contents of FFE80004. The next part of the command line, ? 22222220, tells the monitor to display the original contents of FFE80008, before assigning the new value "22222220" to it. The 33333330 tells the monitor to assign the value "33333330" to memory location FFE8000C. Finally, the q causes the monitor to exit to the command level after the contents of FFE80010 are displayed.

## Special Monitor Commands

### Address Increment/Decrement Command

By preceding the command with a **+** or **-**, you can cause the address display to increment or decrement to the next location.

While traversing a range of addresses, type a **+** or **-** to change direction after the program displays the content and waits for input.

For example:

```
>1 0 
00000000 00000000 ? 
00000004 00000001 ? 
00000008 00000002 ? - (you enter a minus sign) 
00000004 00000001 ?  (contents of previous location are displayed)
00000000 00000000 ? +  (after decrement, you enter a plus sign)
00000004 00000001 ?  you increment again
00000008 00000002 ?  (you increment again)
```

### The **^x** Command — Sun-3/400 series only

A set of commands that are prefaced with the caret (^) character access the second level of the PROM monitor command menu.

Entering the ^ character and then the **x** key displays the following MC68030 (Sun-3/400 series) registers:

The Translation Control Register in the PMMU

The CPU Root Pointer (both Status Long Word and Table Address)

The System Enable Register

The Interrupt Register

The Bus Error Register

Viewing these registers gives a quick view of the current CPU status as opposed to the monitor **x** command, which displays status that was stored during an exception error.

### The **^t** Command

Entering the ^ character and then the **t** key, followed by a virtual address, displays the physical address to which that address is mapped, along with a detailed description of all the bits in the page table entry, the segment and page RAM addresses, and what space they are in.

For example, entering



>^t 1000 Return

results in this display:

```
Virtual Addr 1000 is mapped to Physical Addr 1000
Context = 0x0, Seg Map = 0x0, Page Map = 0xC0000000.
Page 0 has these attributes:
```

```
Valid bit   = 0
Permission  = 1
No Cache    = 0
Type        = 0
Access bit  = 0
Modify bit  = 0
```

Entering ^t with no arguments provides information with reference to virtual address 0x00.

#### The ^i Command

Entering the ^ character and then the i key, followed with a command, displays compilation information for the system firmware. It includes the date, host name and build directory path. For example:

```
Compiled at 6/7/87 on hostname in /directory_name
```

#### The ^c Command

**^c** *source destination n*

Entering the ^ character and then the c key, followed with the parameters shown, causes a block of *n* length to be copied from *source* to *destination* address, byte by byte. There is enough delay to copy to EEPROM also.

#### The ! Command

Entering an exclamation point executes the last monitor command you entered again.

#### Regular Monitor Commands

##### Monitor **a** Command

**a** *register\_number action*

The **a** command provides access to the CPU's address registers. *register\_number* may be a value from 0 to 7 inclusive. The default value is 0. Register\_number 7 accesses A7, the system stack pointer. To see the user stack pointer, use the **r** command.

It is important to note that it is *not* possible to display the state of the processor at all times. In particular, processor state is only observable after:

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`).
- You have manually "broken" into the monitor mode (by typing the L1-a sequence on a Sun keyboard or Break on a dumb terminal's keyboard).

Read the section entitled *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **a**.

#### Monitor **A** Command—Sun-3/400 Series

For the Sun-3/400 series, the uppercase **A** command displays these physical and virtual addresses:

| <i>Device</i> | <i>Virtual Address:</i> | <i>Physical Address:</i> |
|---------------|-------------------------|--------------------------|
| Keybrd/Mouse  | 0x:fef00000             | 0x:62000000              |
| Serial Port   | 0x:fef02000             | 0x:62002000              |
| EEPROM        | 0x:fef04000             | 0x:64000000              |
| TOD           | 0x:fef06000             | 0x:64002000              |
| Mem Err Reg   | 0x:fef09000             | 0x:61001000              |
| Int Reg       | 0x:fef0b400             | 0x:61001400              |
| P4 DAC        | 0x:fef0c000             | 0x:50200000              |
| ie Ethernet   | 0x:fef08000             | 0x:65000000              |
| ECC ENA Reg   | 0x:fef14000             | 0x:6a1e0000              |
| Sys ENA Reg   | 0x:fef16000             | 0x:61000000              |
| Bus ERR Reg   | 0x:fef18400             | 0x:61000400              |
| IDPROM        | 0x:fef1cc00             | 0x:61000c00              |
| P4 Video Reg  | 0x:fef1e000             | 0x:50300000              |
| VIDEOMEM_BASE | 0x:fef20000             | 0x:50400000              |
| P4 Overlay    | 0x:fef20000             | 0x:50400000              |
| P4 ENA Plane  | 0x:fef40000             | 0x:50600000              |
| IOMAPPER      | 0x:fef66000             | 0x:60000000              |
| CACHETAGS     | 0x:fefc0000             | 0x:68000000              |
| CACHEDATA     | 0x:fefd0000             | 0x:69000000              |
| IO CACHE TAGS | 0x:fef6c000             | 0x:6c000000              |
| IO CACHE DATA | 0x:fef6e000             | 0x:6c002000              |

#### Monitor **b** Command

##### **b** ? or **b!** *boot\_device path argument\_list*

The **boot** command loads and executes the SunOS operating system, an EEPROM-specified program, or a user-specified program. The boot program can be loaded from the default device, the device specified in the EEPROM, or the boot device specified in the command argument. A *boot\_device* is a secondary storage device (disk, Ethernet or tape) that contains the program to be loaded and executed.

If the diagnostic switch on the back of the system is in the NORM position, the value in EEPROM address 0x18 is *not* equal to 0x12, and the boot command is entered without arguments, the system will boot the SunOS operating system, using the following default boot device polling sequence.

1. Xylogics Disk (xy), (xd), (xt).
2. SCSI Disk (sd), (st).
3. Ethernet (ie), (le).

If the EEPROM value at address 0x18 is equal to 0x12, the system will boot the SunOS operating system from an EEPROM-specified device. The boot device is specified in locations 0x19 through 0x1D, inclusive, of the EEPROM. Refer to the **q** command for information on how to open and modify these EEPROM



locations.

When the diagnostic switch is in the DIAG position and command **b** is entered by itself, the system will boot an EEPROM-specified program from an EEPROM-specified device. In this case, the boot path is specified in locations 0x28 through 0x50, inclusive, of the EEPROM; the boot device is specified in locations 0x22 through 0x26, inclusive, of the EEPROM. If the boot attempt fails, the user is returned to the monitor's command level.

In order to boot from a specific device, the **b** command must be followed with a boot device abbreviation, such as those shown below. Enter **b ?** to view the boot device identifier arguments that your PROM monitor will accept.

**b** *device (controller, unit, partition) path argument\_list*

You must surround *controller*, *unit*, and *partition* with parentheses, and place a comma after each entry. To invoke the default values, simply enter:

**b** *device* (,,)

*device* may be one of the following:

- fd — Floppy Disk
- xd — Xylogics 7053 Disk
- xy — Xylogics 450/451 disk
- sd — SCSI disk
- ie — Intel Ethernet
- st — SCSI tape
- xt — Xylogics 472 Tape
- mt — Tape Master 9-Track Tape

*controller* stands for the Controller Number, referring to the tape or disk controller board. The default is 0.

*unit* refers to Unit Number, meaning disk number. The default is 0.

*partition* is the partition number of the boot device. The default is 0.

*path* is the path and filename of the program to boot.

*argument\_list* is the list of arguments for the boot program. Up to seven optional arguments (which are passed to the boot program) may follow the path argument.

If you do not want the system to be reset prior to booting, you must insert **!** before the device identifier argument.

**b** *ie*(0,0,1)/stand/video.diag -t

The boot command shown above would boot the video diagnostic from the /stand directory over the Ethernet. Because the **!** argument does *not* precede the device identifier argument, the system is reset before the booting process begins. Note that one optional argument (-t) is passed on to program video.diag.

Monitor **c** Command**c** *virtual\_address*

The `continue` command resumes execution of an interrupted program. You can specify the virtual address of the instruction to be executed when the program restarts.

By default, the program resumes execution at the address pointed to by the program counter.

**NOTE** *This command is helpful if you should use the L1-A sequence and decide you did not want to abort the operating system.*

Monitor **d** Command**d** *register\_number action*

The `data register` command provides access to the CPU's data registers. *register\_number* can be a value from 0 to 7 inclusive. The default value is 0. If you do not specify a register, this command displays the data registers one-at-a-time, in ascending order.

It is important to note that it is *not* possible to display the state of the processor at all times. In particular, processor state is only observable after:

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`)
- You have manually "broken" into the monitor (by typing `L1-a` on the console's keyboard or **break** on the "dumb" terminal's keyboard).

Read the previous section, *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **d**.

Monitor **e** Command**e** *virtual\_address*

The `display/modify memory` command displays and/or modifies the content of one or more virtual addresses in *word* mode (i.e. each virtual address will be treated as a 16-bit unit). That is, the content of one or more words can be *displayed*, *modified* or, both *displayed* and *modified*.

See the previous section *Displaying and Modifying Memory* for a description of this command's syntax. Use the letter **e** in place of the word *command* in the description.

Monitor **f** Command**f** *start\_virtual\_address end\_virtual\_address pattern size*

The `block write` command writes the *pattern* you enter into each byte, word or long word in the range of virtual addresses you specify with the *start\_virtual\_address* and *end\_virtual\_address* arguments. By default, if the final, memory-cell-size argument is not present, bytes are written. Arguments *start\_virtual\_address*, *end\_virtual\_address* and *pattern* are required while *size* is optional. The possible values for *size* are **b** (8-bit byte), **w** (16-bit word) or **l** (32-bit long word).



## Monitor **F** Command—Sun-3/400 Series

For the Sun-3/400 series, the **F** command flushes the ATC Cache on the MC68030 CPU.

## Monitor **g** Command

**g** *vector argument*  
or

**g** *virtual\_address argument*

When you use the **goto** command, you may go to (jump to) a *pre-determined*, *user-specified* or *default* routine. If a pre-determined or default routine is invoked, optional arguments *vector* (a hexadecimal number) and *argument* (a string) are passed to the routine to be executed.

In its other form, the argument *virtual\_address* is used to indicate the virtual address of a *user-specified* routine, and the optional *argument* is passed along to that routine. If you do not supply the *vector/virtual\_address* argument, the value in the Program Counter is used.

In order to set up a *pre-determined* routine, a user program has to set variable *\*romp->v\_vector\_cmd* equal to the virtual address of the routine. Variable *\*romp->v\_vector\_cmd* must be set prior to executing the **g** command. Pre-determined routines may or may not return to the monitor.

The *default* routine, defined by the monitor, simply prints the user-specified *vector* argument according to the user-specified format (given in *argument*) before returning to the monitor. The only allowable formats are *%x* and *%d*. Format *%x* prints argument *vector* as a hexadecimal number while format *%d* prints argument *vector* as a decimal number.

## Monitor **h** Command

**h** or **?**

The **help** command brings up a Help display that describes the basic monitor commands and their argument(s). An example of the help menu is shown at the beginning of this chapter.

If the help menu on your system has numbered entries, you may enter the number next to the command, and then **Return** to obtain further information about that command.

## Monitor **i** Command — Sun-3/200 or Sun-3/400 series

**i** *c icache\_data\_offset*

The **modify cache data RAM** command displays and/or modifies the contents of one or more of the cache data addresses. For Sun-3/400 series systems, you may add a *c* after entering *i*, to specify Central Cache. Adding a second *i* specifies I/O Cache.

Read the previous section *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter *i*.

### Monitor **j** Command — Sun-3/200 or Sun-3/400 Series

#### **j** *cache\_tag\_offset*

The `modify cache tag RAM` command displays and/or modifies the contents of one or more of the cache tag addresses. For Sun-3/400 Series systems, you may add a `c` to specify Central Cache, or an `i` to specify I/O Cache.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **j**.

### Monitor **k** Command

#### **k** *reset\_level*

The `reset` command performs various levels of system resets. It can also be used to display the system's banner. This command accepts one optional argument. Entering **k 0** resets the VME-bus, interrupt register and video monitor (the Low-Level Reset).

Entering **k 1** invokes a Software Reset.

Entering **k 2** invokes a Power-On Reset (Hard Reset).

Finally, entering **k b** displays the banner on the video monitor. The default value of the argument is `0`.

### Monitor **l** Command

#### **l** *virtual\_address*

The `modify long words of memory` command displays and/or modifies the contents of one or more virtual addresses in *long* word mode. Each virtual address is treated as a 32-bit unit (long word). Read "Displaying and Modifying Memory" for details on how to use this command. Replace the word *command* in the description with the letter **l**.

### Monitor **m** Command

#### **m** *a b virtual\_address*

The `modify segment table` command displays and/or modifies the contents of one or more of the Segment Table entries. For Sun-3/400 series systems, you may specify `a` for translation table TIA, or `b` for translation table TIB. If you do not specify the table, TIA is displayed by default.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **m**.



## Monitor **n** Command — Sun-3/200 and 3/400 series only

### **n c i** *cache\_command*

The **control cache** command globally controls the cache. Entering **n d** *disables* the cache (Central Cache for Sun-3/400 series systems). Entering **n e** *enables* the cache (Central Cache for Sun-3/400 series systems). Finally, entering **n i** *invalidates* the cache (Central Cache for Sun-3/400 series systems).

For Sun-3/400 series systems, enter

**ni d, e, or i**

to perform the action on the I/O Cache. To select the Central Cache, enter

**nc d, e, or i**

## Monitor **o** Command

### **o** *virtual\_address*

The **modify bytes of memory** command displays and/or modifies the content of one or more virtual addresses in *byte* mode. Each virtual address is treated as an 8-bit unit (byte). Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **o**.

## Monitor **p** Command

### **p** *virtual\_address*

The **modify page table** command displays and/or modifies the contents of one or more of the Page Table entries. For Sun-3/400 series systems, the virtual address determines whether the page displayed is from the page table or from the I/O Mapper RAM. A virtual address between 0xff000000 and 0xffffffff searches for the requested page in the I/O Mapper RAM pages.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **p**.

## Monitor **q** Command

### **q** *eprom\_offset* or **q \***

The **modify bytes of EEPROM** command displays and/or modifies the configuration information within the EEPROM in *byte* mode. This command works similarly to the memory commands discussed previously, except that the modified addresses are offset, and the changes you make remain when you power-down the workstation. Read the previous section, *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **q**. Refer to the *Sun-3 EEPROM Layout* chapter of the for information *PROM User's Manual* on what parameters are stored at which EEPROM addresses.

**q\***—Version 2.4 and later Boot PROM Only

In the Version 2.4 Boot PROM, you may enter an asterisk instead of an EEPROM offset value as an argument, and the **q** command will erase the entire EEPROM.

Monitor **r** Command**r** *register\_symbol*

The miscellaneous register command displays and/or modifies the contents of the CPU's miscellaneous registers. You may enter any of the symbols shown on the table below to specify registers. For example, if you enter the following, you will display the state of the MC68030 Cache Control register:

```
>r cc
```

Table 2 *Miscellaneous Registers for the 68020*

| <i>Symbol</i> | <i>Name</i>               |
|---------------|---------------------------|
| IS            | Interrupt Stack Pointer   |
| MS            | Master Stack Pointer      |
| US            | User Stack Pointer        |
| SF            | Source Function Code      |
| DF            | Destination Function Code |
| VB            | Vector Base               |
| CA            | Cache Address Register    |
| CC            | Cache Control Register    |
| CX            | Context Register          |
| SR            | Status Register           |
| PC            | Program Counter           |

Table 3 *Miscellaneous Registers for the 68030*

| <i>Symbol</i> | <i>Name</i>            |
|---------------|------------------------|
| SS            | System Stack           |
| US            | User Stack             |
| VB            | Vector Base Register   |
| PC            | Program Counter        |
| SR            | CPU Status Register    |
| CC            | Cache Control Register |
| CA            | Cache Address Register |

It is important to note that it is *not* always possible to display the registers. They may be observed only after

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`) or
- You have manually "broken" into the monitor (by typing `L1-a` on the Sun keyboard or `Break` on a "dumb" terminal's keyboard).

Read the previous section *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **r**.



### Monitor **R** Command—(Sun-3/400 Series only)

The uppercase **R** command reads the System Enable, Interrupt, Bus Error, I/O Mapper and Memory Error registers in the MC68030. Addresses of those registers are as follows:

| <i>Register</i> | <i>Address</i> |
|-----------------|----------------|
| System Enable   | 0xa100         |
| Interrupt       | 0x81           |
| Bus Error       | 0xe3           |
| I/O Mapper      |                |
| Memory Error    | 0x007feab8     |

### Monitor **s** Command

**s** *number*

The *modify* command sets or displays the address space to be used by subsequent memory access commands. The processor function codes decode the address spaces. Argument choices represent the function codes:

Table 4 *Function Code Values*

| Value | Address Space                                |
|-------|----------------------------------------------|
| 0     | Reserved (don't use)                         |
| 1     | User Data                                    |
| 2     | User Program                                 |
| 3     | Control Space (Reserved in Sun-3/400 Series) |
| 4     | Reserved (don't use)                         |
| 5     | Supervisor Data                              |
| 6     | Supervisor Program                           |
| 7     | Supervisor CPU Space                         |

If you do not enter a function code number, the current setting (either 1 or 5) is displayed, and entry of the monitor **o** command, for example, would cause the program to look for the specified address in either user or supervisor data space. Conversely, if you reset the function code number, you could query registers in the space represented by the number entered. For example, entering

```
>s 3
```

would allow you to read the bus error or system enable register, which are located in Control Space.

### Monitor **T** Command—Sun-3/400 Series

Entering and uppercase **T** for the Sun-3/400 series translates the virtual address entered after the command to its corresponding physical address, similarly to the **^t** command for other Sun-3 systems. If you entered

```
>t 12345678
```

for example, the display might look like this:

Virtual Addr 0x12345678 is mapped to Physical Addr 0xfef12345  
 TIA Entry = 0x7fa400, TIB Entry = 0x7fa60a PTE = 0x59

Page 0x1a has these attributes:

No Cache = 1  
 Modify bit = 1  
 Access bit = 1  
 Write Protect bit = 0  
 Descriptor Type = 1

Monitor **u** Command

**u** *port options baud\_rate*  
 or

**u** *echo*  
 or

**u** *uvirtual\_address*

The *input/output* command configures the input and output devices and their characteristics or displays the current input and output device set-up.

The **u** command requires arguments to specify from which device(s) you want the system to expect input or which device(s) will display output.

If you do not enter an argument after the **u** command, the program will display the current settings. If no serial port is specified when changing baud rates, the baud rate of the current input device is changed. The default serial port baud rate is 9600 for both ports during a normal boot. During a diagnostic boot, defaults are 9600 baud for Port A and 1200 for Serial Port B.

Upon normal power-up (diag switch is in NORM position), the default console input device is the Sun keyboard, unless the EEPROM has specified another default input device. If the keyboard is unavailable, the system looks to serial port A for for input.

The default console output device is the Sun monitor (subject to change through EEPROM programming). If the workstation has a color monitor and a color board is unavailable, the program will look for a monochrome monitor as an output device.

You may alter the existing I/O settings while you are in the monitor mode, using the commands listed below; however, the default settings will be reinstated when the system is power cycled.



**u Command Arguments:**

The *port* argument can be one of the following:

Table 5 *Port Arguments*

| Port Argument | Device        |
|---------------|---------------|
| a             | Serial Port A |
| b             | Serial Port B |
| k             | Keyboard      |
| s             | Screen        |

The *options* arguments are:

Table 6 *Option Arguments*

| Option Argument | Meaning                           |
|-----------------|-----------------------------------|
| i               | input                             |
| o               | output                            |
| u               | UART                              |
| e               | input echoed to output            |
| ne              | input <i>not</i> echoed to output |
| r               | reset specified serial port       |

The *baud\_rate* argument specifies baud rate of the serial port being discussed. The *virtual\_address* argument specifies the virtual address of the UART (Universal Asynchronous Receiver/Transmitter).

Following are examples of port and options arguments:

- Enter **u aio** or **u bio** to select serial port A or B as the input and output device.
- Enter **u ai** or **u bi** to select serial port A or B for input only.
- Enter **u ao** or **u bo** to select serial port A or B for output only.
- Enter **u k** to select the keyboard for input.
- Enter **u ki** to select the keyboard for input.
- Enter **u s** to select the screen for output.
- Enter **u so** to select the screen for output.
- Enter **u ks, sk** to select the keyboard for input and the screen for output.
- Enter **u abaud rate** or **u bbaud rate** to set the serial port speed.
- Enter **u e** to cause the output to echo the input.
- Enter **u ne** to cause the output *not* to echo the input.
- Enter **u address** to set the serial port virtual address.

A *previously mapped* UART can also be used for input and/or output. Command **u uvirtual\_address''**, where *virtual\_address* is the virtual address of a previously mapped UART, changes the virtual address of the UART. Do not enter a space between the second **u** and the virtual address.

Monitor **v** Command**v** *start\_virtual\_address end\_virtual\_address size*

The display memory block command displays the contents of each byte, word or long word in the range of virtual addresses specified by *start\_virtual\_address* and *end\_virtual\_address*. The *word\_size* argument is optional; the default is to display memory in byte format. In this format, sixteen consecutive bytes are displayed on each line. In word format, eight consecutive words appear on each line.

If long word format is enabled, four consecutive long words appear on each line. To the far right of each line, the character corresponding to each byte is also shown. All bytes that contain a non-ASCII code are shown as a period (.). Legal values for *size* are **b** (8-bit byte), **w** (16-bit word), or **l** (32-bit long word).

To terminate the **v** command, press the *space bar*. To "freeze" the display, press any key *except* the space bar. To restart the **v** command, press the space bar again.

Monitor **w** Command**w** *virtual\_address argument*

The set execution vector command allows you to vector to a *pre-determined* or *default* routine. Optional arguments *virtual\_address* and *argument* are passed along to the to-be-executed routine.

In order to set up a *pre-determined* routine, a user program sets the variable *\*romp->v\_vector\_cmd* equal to the virtual address of the *pre-determined* routine. Variable *\*romp->v\_vector\_cmd* must be set prior to executing the **w** command. *Pre-determined* routines may or may not return to the monitor.

The *default* routine, defined by the monitor, simply prints the user-specified *virtual\_address* argument according to the user-specified format (given in *argument*) before returning to the monitor. The only allowable formats are "**%x**" and "**%d**". Format "**%x**" prints argument *virtual\_address* as a hexadecimal number; format "**%d**" prints it as a decimal number.

Monitor **x** Command**x**

The extended test system command invokes the Extended Test Sequence or the Extended Test System, depending on what Boot PROM revision is on the CPU board. See *Chapter 9* for details.

Monitor **y** Command — Sun-3/200 or -3/400 Series**y** *c number*  
or**y** *cache\_section number virtual\_address*

The flush cache command performs a number of *flush* operations on the cache. Depending on what is to be flushed, two or three arguments are required. In order to flush a context, enter command **y** *c number*, where *number* is a valid context number.

Entering **y** *s number virtual\_address* flushes segment *virtual\_address* within context *number*. The argument *number* is a valid context number.



Entering **y p** *number virtual\_address* flushes page *virtual\_address* within context *number*. The argument *number* is a valid context number.

Monitor **z** Command

**z** *number breakpoint\_virtual\_address type len*

Command **z** sets or resets breakpoints for debugging purposes. Optional argument *number* can have values from 0 to 3, corresponding to the processor debug registers DR0 to DR3, respectively. Up to four distinct breakpoints can be specified. If argument *number* is not specified, the monitor will choose a breakpoint number.

The argument *breakpoint\_virtual\_address* specifies the breakpoint address to set.

The argument *type* can have three values: **x** for Instruction Execution breakpoint, **m** for Data Write only breakpoint, and **r** for Data Reads-and-Writes-only breakpoint. The default value for *type* is **x**.

The argument *len* can also have three values: **b**, **w**, and **l**, corresponding to the breakpoint field length of byte, word, and long-word, respectively. The default value for *len* is **b**. Since the breakpoints are set in the on-chip registers, an instruction breakpoint can be placed in ROM code or in code shared by several tasks.

If the argument *number* is specified but the argument *breakpoint\_virtual\_address* is not specified, then the corresponding breakpoint will be reset.

This command without any arguments will display all the existing breakpoints.

## 1. SPARCsystem 330 Self-Tests and Monitor Commands

For the most part, the diagnostic self-test and monitor command descriptions for the SPARCsystem 330 are the same as those in Chapters 11, 12 and 13 of the *PROM User's Manual*. This addendum provides descriptions of commands or tests that differ from those used for Sun-4/110 or Sun-4/200 series systems.

When a terminal is attached to Serial Port A, and the system is in diagnostic mode, the name of each self-test is displayed as the test is executed, as shown on the following pages.

When in normal mode, self-tests for 8 Mbytes of memory may last 45 seconds; a 128 MB memory may take up to 8 minutes. In diagnostic mode, when all of memory is tested, an 8 MB system may take four minutes to complete self-tests, while a 128 MB system may take 56 minutes to execute the self-tests.

### Self-Test Interaction

There are several commands that allow limited interaction with the SPARCsystem self-tests, and they are described below.

- b** Press the **b** (a mnemonic for *burn-in*) key, prior to the display of the ...Completed or Selftest Finished message, to execute the power-up test sequence indefinitely. This option is useful during the manufacturing burn-in stage.

For the Sun-3/80, when the last selftest is finished the message testsAutomaticallycontinuing will be displayed and the self-test will be restarted again, using the System Enable Register read test as the first test. The SCC and terminal I/O tests are bypassed in burn-in mode since operator interaction is required. This looping sequence will continue until an ESCAPE is entered, a reset is done, or the "b" key pressed again to turn burn-in mode off. The burn-in mode can be toggled by successive pressing of the "b" key. Note that this burn-in mode key can be processed during a test that is running normally (no errors), if the test is presently looping on an error encountered, or at the end of selftest when the "Selftest Finished" is displayed and an operator input is requested.

- s** Press the **s** key prior to the display of the ...Completed message to *re-start* the power-up test sequence.

### Space Bar

If one of the power-up tests fails, it will continue to re-execute forever unless interrupted. Press the space bar to terminate the failed test and execute the next power-up test.

### p — Toggle Print Mode

By default, an unsuccessful power-up test prints one error message before entering an infinite scope loop. Once inside of the scope loop, the failing test will *not* print any more messages. If, however, you would like to see test messages while in the scope loop, press the **p** key. Then, to turn the messages back off, press the **p** key a second time. In other words, the **p** command acts like a toggle switch to turn message mode on or off.

### Escape Key

Pressing Esc prior to the SelftestCompleted message skips the remaining power-up tests and warns:



<Warning: Selftests aborted by user>

#### Control-L

Holding down the **[Control]** key while pressing **L** causes the test that is currently running to re-execute forever. Pressing the **[Control-L]** sequence again turns toggle-loop mode off.

#### Control-Q

Holding down the **[Control]** key while pressing **Q** terminates the current test and skips to the next test in the power-up self-test sequence. This feature works on a passing or failing test.

**NOTE** *If EEPROM location 0x17 is set to 0x12, you may press the User Reset Switch at the back of the system to restart the self-tests. The tests will restart regardless of the position of the Diagnostic Switch.*

Figure 7 SPARCsystem 330 Diagnostic Boot-Up Display

```

Boot PROM Selftest
EPROM Checksum Test.
Context Register Test.
Segment Map Write-Write-Read-Read Test.
Segment Map Address Test.
Segment Map 3-Pattern Test.
Page Map Write-Write-Read-Read Test.
Page Map Address Test.
Page Map 3-Pattern Test.
Software Traps Test: levels 0x80 -> 0xFF
Interrupt Register Test
Software Interrupts Test.
TOD Interrupt Test.
<Probing for a P4 Video Board>
(For Color systems, refer Video tests change as shown on tables that follow
Video Memory Write-Write-Read Test. 32-bit (black and white)
Video Memory Address Test. 32-bit (black and white)
Video Memory 3-Pattern Test. 32-bit (black and white)
<Probing for Main Memory>
<Found 0x00000008 MB of Main Memory>
Limited Memory Test: End of Megabyte 0x00000008. Tests Complete.
MMU Read Access/Modified Bits Test.
MMU Write Access/Modify Bit Test.
MMU Write to Write-Protected Page Test.
MMU Read From Write-Protected Invalid Page Test
MMU Read Writable Invalid Page Test.
MMU Write To Write-Protected Invalid Page Test.
MMU Write To Writable Invalid Page Test.
Main Memory Non Existent Physical Address Read Timeout Test.
Main Memory Invalid Physical Address Read Timeout Test.
Main Memory Invalid Physical Address Write Timeout Test.
Control Space Timeout Test.
Range Error Test
Size Error Test.

```

The Diagnostic Display is continued on the next page.



Figure 8 *Diagnostic Boot-Up Display - Continued*

```

Parity No Fault Test.
Parity Error Detection and Trap Test.
Cache Tag RAM Write-Write-Read-Read Test.
Cache Tag RAM Address Test.
Cache Tag RAM 3-Pattern Test.
Cache Data RAM Write-Write-Read-Read Test.
Cache Data RAM Address Test.
Cache Data RAM 3-Pattern Test.
<Exiting boot state>
<Successfully exited boot state>
Cache Read Hit (Addr Match,Valid) Test
Cache Read Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Read Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Read Miss (Addr Match,Invalid,Cacheable) Test
Cache Read Miss (No Addr Match,Valid,Cacheable) Test
Cache Read Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Write Hit (Addr Match,Valid,M=1) Test
Cache Write Hit (Addr Match,Valid,M=0) Test
Cache Write Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Write Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Write Miss (Addr Match,Invalid,Cacheable) Test
Cache Write Miss (No Addr Match,Valid,Cacheable) Test
Cache Write Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Flush Context (Cntx Match) Test
Cache Flush Context (Cntx No Match) Test
Cache Read Double Hit (Addr Match,Valid) Test
Cache Read Double Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Read Double Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Read Double Miss (Addr Match,Invalid,Cacheable) Test
Cache Read Double Miss (No Addr Match,Valid,Cacheable) Test
Cache Read Double Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Write Double Hit (Addr Match,Valid,M=1) Test
Cache Write Double Hit (Addr Match,Valid,M=0) Test
Cache Write Double Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Write Double Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Write Double Miss (Addr Match,Invalid,Cacheable) Test
Cache Write Double Miss (No Addr Match,Valid,Cacheable) Test
Cache Write Double Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Successive Write Double Hit (Addr Match,Valid,M=1) Test
Cache Successive Write Double Hit (Addr Match,Valid,M=0) Test

```

This display is continued on the following page.



Figure 9 Diagnostic Boot-Up Display - Continued

```

Atomic Load/Store Hit (Addr Match,Valid) Test
Atomic Load/Store Miss (Addr Match,Invalid) Test
Atomic Load/Store Hit (Addr Match,Valid,Write Protected Page) Test
Atomic Load/Store Miss (Addr Match,Invalid,Write Protected Page) Test
<Re-entering boot state>
<Successfully re-entered boot state>
VME Loopback and DVMA Test
VME Loopback and DVMA Read Timeout Test
VME Loopback and DVMA Write Timeout Test

END OF SELFTEST # 0x00000000 (SELFTEST PASSED) .
#PASSED = 0x00000000, #FAILED = 0x00000001

<Sizing Main Memory>
<Found 0x00000008 MB of Main Memory>
<Initializing Memory...>
<Turning Parity Checking ON>
Main Memory Test: Megabyte 0x00000008. Tests Complete.
<Found 0x00000008 MB of Main Memory>
<Initializing Memory...>

Selftest Completed.

```

Type a key within 10 seconds to enter Extended Test System (e for echo mode).

If the system contains a CG6 board, the self-tests function the same as the Video tests shown previously, and the display looks like this:

```

<Probing for a P4 Video Board>
Video Memory Write-Write-Read-Read Test. 32-bit - Color Plane.
Video Memory Address Test. 32-bit - Color Plane.
Video Memory 3-Pattern Test. 32-bit - Color Plane.

```

If the system contains a CG8 board, the self-tests function the same as the Video tests shown previously, and the display looks like this:

```

<Probing for a P4 Video Board>
Video Memory Write-Write-Read-Read Test. 32-bit - Overlay Plane.
Video Memory Address Test. 32-bit - Overlay Plane.
Video Memory 3-Pattern Test. 32-bit - Overlay Plane.
Video Memory Write-Write-Read-Read Test. 32-bit - Enable Plane.
Video Memory Address Test. 32-bit - Enable Plane.
Video Memory 3-Pattern Test. 32-bit - Enable Plane.
Video Memory Write-Write-Read-Read Test. 24-bit - Color Plane.
Video Memory Address Test. 24-bit - Color Plane.
Video Memory 3-Pattern Test. 24-bit - Color Plane.
P4 Color Map Test.

```

If the Diagnostic Switch is in the *diagnostic* position, the power-up tests executed successfully and the user did *not* enter a character on either of the *terminal* keyboards within the initial ten second period, the following display will appear on the *workstation's* screen. Again, note that you have a *second* ten-second



opportunity to invoke the additional non-power-up tests by pressing *any* key on the workstation's keyboard, except the **(Esc)** key. Pressing the **(Esc)** key in this situation will terminate the delay and invoke the PROM monitor program. Providing two opportunities to enter the Extended Test System will give the user the choice of interacting with either an RS232 terminal or the workstation's keyboard and video monitor.

If you do not respond to either of the *ten-second* delays, an attempt will be made to boot an EEPROM-specified program from the Diagnostic boot path area of the EEPROM. If this attempt should fail, the monitor program is invoked.

### Successful Self-Test Display

The test names actually appear on the screen sequentially as each test is performed. The previous example shows the display on the the terminal (connected to Serial Port A at 9600 baud or Port B at 1200 baud) screen after all the self-tests are successfully completed.

**NOTE** *If no terminal is connected to the serial port, you will not be able to interact with the self-tests, and you won't see the Diagnostic Boot-Up Display. If self-test is successful, the console display prompts will offer choices described on the following page.*

Selftest Completed Successfully



Sun Workstation, Model Sun-4/300 Series,  
Sun-4 Keyboard

ROM Rev \_\_, \_\_ MB memory installed, Serial # \_\_\_\_  
Ethernet address \_\_:\_\_:\_\_:\_\_:\_\_:\_\_

Type a character within 10 seconds to enter Extended Test System.

If you do not press a key on the console keyboard, the Boot PROM program checks parameters stored in the EEPROM that tell it to do one of the following:

- ☐ Boot a specific (diagnostic) program from a specific device
- ☐ Drop into the PROM monitor mode.

If you press a key, the Extended Test System menu is offered. Refer to the SPARCsystem 330 extended test system addendum at the end of this chapter.

### To Read the CPU Board LED Table

The following table provides a brief interpretation of the patterns displayed by the LED indicators on the edge of the CPU board. For vertical installations, the LED depicted on the left in this table is located at the top of the display; the LED on the right is at the bottom of the display.







## The SPARCsystem 330 PROM Monitor

The boot PROM stores a program known as the "monitor" that contains self-tests and controls system operation during boot-up until the SunOS kernel takes over. The monitor program is invoked when you use the halt SunOS, and is identified with the ">" prompt.

The PROM monitor program provides commands that perform a variety of tasks, such as booting from alternate devices, changing the console output to a serial port, and so on.

Access the monitor as described in the Manual, *PROM User's* then enter **h** to view a "Help" table of monitor commands that looks something like this:

| Monitor                                                                | REV:1                                                   | date | Help Menu |
|------------------------------------------------------------------------|---------------------------------------------------------|------|-----------|
| 1 b                                                                    | Boot a program.                                         |      |           |
| 2 k                                                                    | Reset all or part of the machine or display the banner. |      |           |
| 3 u                                                                    | Initialize the input and output devices.                |      |           |
| 4 c/g/w                                                                | Resume or modify program flow.                          |      |           |
| 5 d/r                                                                  | Display and/or modify the registers.                    |      |           |
| 6 o/e/l                                                                | Display and/or modify individual memory locations.      |      |           |
| 7 ^c/f/v                                                               | Copy, display or modify a block of memory.              |      |           |
| 8 m/p                                                                  | Display and/or modify segment or page table entries.    |      |           |
| 9 q                                                                    | Display and/or modify EEPROM locations.                 |      |           |
| 10 s                                                                   | Display or modify the Address Space Identifier.         |      |           |
| 11 i/j                                                                 | Display and or modify cache data or tag entries.        |      |           |
| 12 n/y                                                                 | Disable, enable, invalidate, or flush the cache.        |      |           |
| 13 ^a/^t                                                               | Display virtual and physical address information.       |      |           |
| ^i                                                                     | Display firmware compilation information.               |      |           |
| ^p                                                                     | Enable or disable parity circuitry.                     |      |           |
| !                                                                      | Re-execute the previously executed command.             |      |           |
| h/?                                                                    | Enter the help system for the basic monitor commands.   |      |           |
| x                                                                      | Enter the Extended Test System.                         |      |           |
| ''option_number''= Additional Help <esc> = Go-To-Previous Menu q= quit |                                                         |      |           |
| Command == >                                                           |                                                         |      |           |

If you need more help concerning any of the Help Menu options, enter the number that precedes the command, and then RETURN.

The "Monitor (8S)" section of the *Commands Reference Manual* also provides information on PROM monitor commands.

The monitor commands described in Chapter 12 of the *PROM User's Manual* are valid for the SPARCsystem 330, with the minor changes described below.

When using the **i**, **j** or **n** or **y** commands, you must specify the type of cache on which you want the operation performed. For example, you may enter the monitor **j** command, followed with **cpu** and the central processor cache will be opened for display or modification. If you replaced **cpu** with **io**, the I/O cache would be displayed or modified:



### j *cpu cache\_offset*

Refer to the "Displaying and Modifying Memory" section of the *PROM User's Manual* for more information on the use of these commands.

### Monitor x Command

Add the following SPARCsystem 330 registers to the Processor Register table in the *PROM User's Manual* Sun-4 PROM Monitor Command descriptions:

| Register Number | Register Name           |
|-----------------|-------------------------|
| 0x80 - 0x87     | g0,g1,g2,g3,g4,g5,g6,g7 |
| 0x88 - 0x8d     | PSR,PC,nPC,WIM,TBR,Y    |
| 0x8e - 0xae     | FSR,f0,...,f31          |

### Monitor s Command

For SPARCsystem 330 CPU boards, you may enter the following Address Space Identifiers (ASI) after the s command. If you do not enter an (ASI) value, the current value will be displayed.

| Name                   | ASI (hex) |
|------------------------|-----------|
| Control Space          | 2         |
| Segment Map            | 3         |
| Page Map               | 4         |
| Block Copy             | 5         |
| User Instruction       | 8         |
| Supervisor Instruction | 9         |
| User Data              | a         |
| Flush Page             | d         |
| Flush Context          | e         |
| Flush User             | f         |

### Monitor t Command

For the SPARCsystem 330, the t command displays or modifies the content of one or more region table entries. Refer to the *PROM User's Manual* for more information on using and modifying memory locations.

### Monitor v Command

The "view" command, described in the *PROM User's Manual* For the SPARCsystem 330, pressing the space bar terminates the command. To freeze the display before it scrolls off the screen, press any key *other than the space bar*. To restart the v command, press any key except the space bar.

If you enter two asterisks (\* \*) at the end of the v command string (which includes the low and high virtual address and byte, word or long word specification), the view command will continue to read each location until you press the space bar. If you add only one asterisk, the content of each location will be displayed once.



**Monitor ^a Command**

For SPARCsystem 330 CPU boards, entering the ^ character followed by an a displays a list of devices and their corresponding physical and virtual addresses.

**Monitor ^c Command**

For SPARCsystem 330 CPU boards, entering the ^ character followed by an c, a source and destination virtual address, and the number of bytes, copies the content of those locations to another location.

**Monitor ! Command**

Entering an exclamation point after the PROM monitor prompt on a SPARCsystem 330 repeats the last executed basic monitor command.

**Identifying A Faulty Memory Module**

In the event a memory error is detected during the power-on self tests, the faulty SIMM will be identified by its "U Number" designation.

Additional memory tests are available in the PROM monitor Extended Test System. After entering the "Help" table shown at the beginning of the "The PROM Monitor" subsection, select x and enter the extended test system. Most of the extended tests are the same as those documented in Chapter 13 of the *PROM User's Manual*. Chapter 13 describes the extended test system user interface. You may enter command lines, or simply make menu selections. You need only enter the letters shown in upper case on the extended test menus.

**NOTE** *Entering an exclamation point displays the last five command lines you have entered. To repeat one of those command lines, simply enter the appropriate command line number.*

The tests that differ from those described for the Sun-4/200 and Sun-4/110 series of workstations are described in the next section.

**SPARCsystem 330 Extended Tests**

```
Monitor REV:some number Date Main Menu
```

```
All           All Test Sequence.
Default       Default Test Sequence.
Boot         Boot Paths Menu.
Ethernet     Ethernet Menu.
Keyboard     Keyboard and Mouse Menu.
Memory       Memory Menu.
Serial       Serial Ports Menu.
Video       Video Menu.
Color       Color Map Menu.
Options     Set global flags/options.
```

```
? = Help <esc> = Return-To-Previous-Menu l = [n] = Repeat-Command-Line-n-Times q= Quit
Command == >
```

The first Main Menu choice, All, brings up the tests shown below. The Default selection runs some of, but not all of these tests. Note that the Ethernet loopback connector, as well as the within-channel external loop-back cables for the keyboard, mouse, Serial Port A, and Serial Port B must be installed prior to running the All sequence. Contact Sun customer support to obtain a



loopback connector kit. The pin assignments for the various connectors appear in *The PROM User's Manual*, with the exception of the Ethernet loopback connector. The pin assignments for the Ethernet connector are:

Figure 10 *Ethernet Loopback Connector*

| <i>From Pin</i> | <i>To Pin</i> |
|-----------------|---------------|
| 3               | 5             |
| 10              | 12            |
| 13              | 14            |

These tests are performed when the All test sequence is chosen from the Extended Test Main Menu for a black and white system:

```

All Test Sequence

a. Ethernet Local Loop Back Test
b. Ethernet Encoder Loop Back Test
c. Ethernet External Loop Back Test
d. Keyboard Register 12 Test
e. Mouse Register 12 Test
f. Keyboard Transmit Test
g. Mouse Transmit Test
h. Keyboard Internal Loop Back Test
i. Mouse Internal Loop Back Test
j. Keyboard External Loop Back Test
k. Mouse External Loop Back Test
l. Main Memory Address Test/Byte Mode
m. Main Memory Constant Pattern Test/Word Mode
n. Main Memory Fill Utility/Long Mode
o. Main Memory Check Utility/Long Mode
p. Serial Port A Register 12 Test
q. Serial Port B Register 12 Test
r. Serial Port A Transmit Test
s. Serial Port B Transmit Test
t. Serial Port A Internal Loop Back Test
u. Serial Port B Internal Loop Back Test
v. Serial Port A External Loop Back Test
w. Serial Port B External Loop Back Test
x. Video Address Test/Byte Mode
y. Video Constant Pattern Test/Word Mode
z. Video Fill Utility/Long Mode
aa. Video Check Utility/Long Mode
Cmd=>

```



If the SPARCsystem 330 has a color monitor, these tests are also performed:

```

Overlay Plane Address Test/Byte Mode
Overlay Plane Constant Pattern Test/Word Mode
Overlay Plane Fill Utility/Long Mode
Overlay Plane Check Utility/Long Mode
Enable Plane Address Test/Byte Mode
Enable Plane Constant Pattern Test/Word Mode
Enable Plane Fill Utility/Long Mode
Enable Plane Check Utility/Long Mode
Color Plane Address Test/Byte Mode
Color Plane Constant Pattern Test/Word Mode
Color Plane Fill Utility/Long Mode
Color Plane Check Utility/Long Mode
Color Map Address Test
Color Map Constant Pattern Test
Color Map Fill Utility
Color Map Check Utility

```

Consult the Sun-4 Extended Test System chapter of the *PROM User's Manual* for descriptions of the tests listed above. The Extended Test System user interface is the same for all Sun-4 systems. The color tests function exactly as the Video tests described in Chapter 13 of the *PROM User's Manual*, except that they test the overlay, enable and color planes rather than the video circuitry. Some SPARCsystem extended tests documented in the *PROM User's Manual* may be labeled "for Sun-4/2xx only" or "for Sun-4/110 only". If the test is identical to one of the SPARCsystem 330 tests, please ignore the label.

## SPARCsystem 330 Initialization

After error-free completion of the power-up self-tests, but before execution of the default boot sequence, the PROM initializes the system. The initialization sequence is:

1. The system enable register is set to normal state.
2. The context register is set to zero.
3. The processor status register is initialized: the CPU is in supervisor mode, window zero is active, level 14 and 15 interrupts are allowed, and traps are enabled. If the floating point processor probe was successful, the Floating Point Unit is enabled.
4. The virtual address of the trap table is written into the trap base register.
5. The window invalid mask register is initialized to 0x2.
6. All page table entries are initialized.
7. Memory is sized.
8. All available main memory is initialized to zero (an unimplemented instruction.)
9. The initial 32 Megabytes of working memory is mapped in ascending order, starting at location zero.



10. One page of RAM is reserved for a stack area and the stack pointer is initialized.
11. A page of RAM is reserved for the PROM monitor global variables.
12. An additional page of RAM is reserved for the monitor's trap vector table and font table.
13. Assuming that they exist, these devices are mapped and initialized:
  - Parity memory registers
  - The EEPROM
  - Other PROMs
  - Ethernet
  - The interrupt register
  - The keyboard and mouse
  - Serial ports
  - Time of day clock
  - P4 black and white board
  - P4 color board
14. The copy of the monitor's trap vector table, which contains the addresses of the trap and interrupt handling routines, is set up.
15. Central processor cache is initialized.
16. Entry points to all support routines provided by the PROM monitor are set up.

---

# Index

## A

automatic boot  
Sun-3, 6

## B

booting procedures  
Sun-3, 8

## C

commands  
SPARCsystem 330 PROM monitor, 109 *thru* 111  
Sun-3/400 PROM monitor, 84 *thru* 102

## D

diagnostics  
serial port, 22  
SPARCsystem 330 power-up, 103 *thru* 108  
Sun-3 power-up, 6, 14, 83  
Sun-3/80 power-up, 9

## E

extended tests  
SPARCsystem 330, 111

## I

initialization  
Sun-3 hardware, 83  
Sun-3/400 Series, 113

## L

L1-a, 8, 90, 97  
LEDs on CPU Board, 7  
Sun-3, 22

## M

monitor  
Sun-3 PROM, 83

## P

power-up display  
Sun-3, 7  
PROM monitor  
Sun-3, 83 *thru* 102  
PROMs  
Sun-3, 83 *thru* 102

## R

remote testing, 13

## S

self-tests  
cache tests, 34, 55  
duration of, 6  
I/O Mapper RAM, 24  
interrupt test, 26  
memory tests, 27  
P4 Video RAM test, 78  
register test, 23, 25  
serial port, 22  
SPARCsystem 330, 103, 104  
Sun-3, 6  
Sun-3/400 series and Sun-3/80 loop menu, 12  
Sun-3/80, 6  
user interaction, 10  
VME and DVMA tests, 54  
VME loopback test, 53  
Sun-3  
monitor commands, 83 *thru* 102  
self-tests, 6 *thru* 83  
Sun-3/80  
self-tests, 82 *thru* 83  
switch  
diagnostics, 9

## T

terminal set-up  
for diagnostics, 10  
testing  
minimum system function, 14





**NAME**

intro – introduction to commands

**DESCRIPTION**

This section describes publicly accessible commands in alphabetic order. Commands of general utility, many with enhancements from 4.3 BSD. Wherever possible, we have incorporated System V versions of commands and utilities into our standard UNIX release. Where a command has both a System V and a BSD version and it has been possible to merge them, we have done so. In some cases, where the System V version is compatible with BSD and offers significant added value, SunOS incorporates that version as its standard.

Pages of special interest have been categorized as follows:

- 1C        Commands for communicating with other systems.
- 1G        Commands used primarily for graphics and computer-aided design.
- 1V        Commands that only have System V versions, or that have separate BSD and System V versions.

These commands either depend upon System V functionality, or have incompatibilities with the corresponding BSD version. They are included in the System V Software installation option. Once installed, they can be found in the directory `/usr/5bin`.

**SEE ALSO**

- Section 8 in this manual for system administration procedures, system maintenance and operation commands, local daemons, and network-services servers.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 6 in this manual for computer games.
- *Getting Started with SunOS: Beginner's Guide*
- *Setting Up Your SunOS Environment: Beginner's Guide*
- *SunView 1 Beginner's Guide*
- *Using the Network: Beginner's Guide*
- *Doing More with SunOS: Beginner's Guide*
- *Programming Utilities and Libraries*

**DIAGNOSTICS**

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program, see `wait` and `exit(2)`. The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status" or "return code," and is described only where special conventions are involved.



## LIST OF COMMANDS

| Name            | Appears on Page     | Description                                               |
|-----------------|---------------------|-----------------------------------------------------------|
| %               | cs(1)               | C shell built-in commands                                 |
| @               | cs(1)               | C shell built-in commands                                 |
| Mail            | mail(1)             | read or send mail messages                                |
| adb             | adb(1)              | general-purpose debugger                                  |
| addbib          | addbib(1)           | create or extend a bibliographic database                 |
| adjacentscreens | adjacentscreens(1)  | connect multiple screens to SunView window driver         |
| admin           | admin(1)            | create and administer SCCS files                          |
| aedplot         | plot(1G)            | graphics filters for various plotters                     |
| alias           | cs(1)               | C shell built-in commands                                 |
| align_equals    | textedit_filters(1) | filters provided with textedit(1)                         |
| ar              | ar(1V)              | create library archives, and add or extract files         |
| arch            | arch(1)             | display the architecture of the current host              |
| as              | as(1)               | Sun-1, Sun-2 and Sun-3, Sun-4 and Sun386i assemblers      |
| at              | at(1)               | execute a command or script at a specified time           |
| atq             | atq(1)              | display the queue of jobs to be run at specified times    |
| atrm            | atrm(1)             | remove jobs spooled by at or batch                        |
| awk             | awk(1)              | pattern scanning and processing language                  |
| banner          | banner(1V)          | display a string in large letters                         |
| bar             | bar(1)              | create tape archives, and add or extract files            |
| basename        | basename(1)         | display portions of pathnames and filenames               |
| batch           | at(1)               | execute a command or script at a specified time           |
| bc              | bc(1)               | arbitrary-precision arithmetic language                   |
| bg              | cs(1)               | C shell built-in commands                                 |
| bgplot          | plot(1G)            | graphics filters for various plotters                     |
| biff            | biff(1)             | give notice of incoming mail messages                     |
| binmail         | binmail(1)          | an early program for processing mail messages             |
| break           | cs(1)               | C shell built-in commands                                 |
| breaksw         | cs(1)               | C shell built-in commands                                 |
| cal             | cal(1)              | display a calendar                                        |
| calendar        | calendar(1)         | a simple reminder service                                 |
| capitalize      | textedit_filters(1) | filters provided with textedit(1)                         |
| case            | cs(1)               | C shell built-in commands                                 |
| cat             | cat(1V)             | concatenate and display                                   |
| cb              | cb(1)               | a simple C program beautifier                             |
| cc              | cc(1V)              | C compiler                                                |
| cd              | cd(1)               | change working directory                                  |
| cdc             | cdc(1)              | change the delta commentary of an SCCS delta              |
| cflow           | cflow(1)            | generate a flow graph for a C program                     |
| checkeq         | eqn(1)              | typeset mathematics                                       |
| checknr         | checknr(1)          | check nroff and troff input files; report possible errors |
| chfn            | passwd(1)           | change password file information                          |
| chgrp           | chgrp(1)            | change the group ownership of a file                      |
| chkey           | chkey(1)            | change your encryption key                                |
| chmod           | chmod(1V)           | change the permissions mode of a file                     |
| chsh            | passwd(1)           | change password file information                          |
| clear           | clear(1)            | clear the terminal screen                                 |
| clear_colormap  | clear_colormap(1)   | clear the colormap to make console text visible           |
| clear_functions | clear_functions(1)  | reset the selection service to clear stuck function keys  |
| click           | click(1)            | enable or disable the keyboard's keystroke click          |
| clock           | clock(1)            | display the time in an icon or window                     |

cluster  
 cmdtool  
 cmp  
 col  
 colcrt  
 coloredit  
 colrm  
 comb  
 comm  
 compress  
 continue  
 cp  
 cpio  
 cpp  
 crontab  
 crtplot  
 crypt  
 csh  
 csplit  
 ctags  
 ctrace  
 cu  
 cut  
 cxref  
 date  
 dbx  
 dbxtool  
 dc  
 dd  
 default  
 defaults\_from\_input  
 defaults\_from\_input  
 defaults\_to\_indentpro  
 defaults\_to\_mailrc  
 defaultsedit  
 delta  
 deroff  
 des  
 df  
 diff3  
 diff  
 diffmk  
 dircmp  
 dirname  
 dirs  
 dis  
 domainname  
 dos2unix  
 dos  
 du  
 dumbplot  
 e  
 echo

cluster(1)  
 cmdtool(1)  
 cmp(1)  
 col(1V)  
 colcrt(1)  
 coloredit(1)  
 colrm(1)  
 comb(1)  
 comm(1)  
 compress(1)  
 csh(1)  
 cp(1)  
 cpio(1)  
 cpp(1)  
 crontab(1)  
 plot(1G)  
 crypt(1)  
 csh(1)  
 csplit(1)  
 ctags(1)  
 ctrace(1)  
 tip(1C)  
 cut(1)  
 cxref(1)  
 date(1V)  
 dbx(1)  
 dbxtool(1)  
 dc(1)  
 dd(1)  
 csh(1)  
 defaultsedit(1)  
 input\_from\_defaults(1)  
 defaultsedit(1)  
 defaultsedit(1)  
 defaultsedit(1)  
 defaultsedit(1)  
 delta(1)  
 deroff(1)  
 des(1)  
 df(1)  
 diff3(1V)  
 diff(1)  
 diffmk(1)  
 dircmp(1V)  
 basename(1)  
 csh(1)  
 dis(1)  
 domainname(1)  
 dos2unix(1)  
 dos(1)  
 du(1V)  
 plot(1G)  
 ex(1)  
 echo(1V)

find the Sun386i cluster containing a file  
 run a shell (or program) using the SunView text facility  
 perform a byte-by-byte comparison of two files  
 filter reverse paper motions from nroff for display  
 filter nroff output for a terminal lacking overstrike  
 alter color map segment  
 remove characters from specified columns within each line  
 combine SCCS deltas  
 display lines in common between two sorted lists  
 compress or expand files, display expanded contents  
 C shell built-in commands  
 copy files  
 copy file archives in and out  
 the C language preprocessor  
 install, edit, remove or list a user's crontab file  
 graphics filters for various plotters  
 encode or decode a file  
 a shell with a C-like syntax  
 split a file with respect to a given context  
 create a tags file for use with vi  
 generate a C program execution trace  
 terminal emulator, telephone connection to a remote system  
 remove selected fields from each line of a file  
 generate a C program cross-reference  
 display or set the date  
 source-level debugger  
 SunView interface for the dbx source-level debugger  
 desk calculator  
 convert and copy files with various data formats  
 C shell built-in commands  
 create or edit default settings for SunView 1  
 update the mouse and keyboard from defaults  
 create or edit default settings for SunView 1  
 create or edit default settings for SunView 1  
 create or edit default settings for SunView 1  
 make a delta (change) to an SCCS file  
 remove nroff, troff, tbl and eqn constructs  
 encrypt or decrypt data using Data Encryption Standard  
 report free disk space on file systems  
 display line-by-line differences between 3 files  
 display line-by-line differences between pairs of text files  
 mark differences between versions of a troff input file  
 compare directories  
 display portions of pathnames and filenames  
 C shell built-in commands  
 object code disassembler for COFF  
 set or display name of the current YP domain  
 convert text file from DOS format to SunOS format  
 SunView window for IBM PC/AT applications  
 display the number of disk blocks used per directory or file  
 graphics filters for various plotters  
 line editor  
 echo arguments to the standard output



|               |                  |                                                            |
|---------------|------------------|------------------------------------------------------------|
| ed            | ed(1)            | basic line editor                                          |
| edit          | ex(1)            | line editor                                                |
| egrep         | grep(1V)         | search a file for a string or regular expression           |
| eject         | eject(1)         | eject floppy disk from an autoeject floppy drive           |
| else          | cs(1)            | C shell built-in commands                                  |
| end           | cs(1)            | C shell built-in commands                                  |
| endif         | cs(1)            | C shell built-in commands                                  |
| endsw         | cs(1)            | C shell built-in commands                                  |
| enroll        | xsend(1)         | send or receive secret mail                                |
| env           | env(1)           | obtain or alter environment variables                      |
| eqn           | eqn(1)           | typeset mathematics                                        |
| error         | error(1)         | categorize compiler error messages                         |
| eval          | cs(1)            | C shell built-in commands                                  |
| ex            | ex(1)            | line editor                                                |
| exec          | cs(1)            | C shell built-in commands                                  |
| exit          | cs(1)            | C shell built-in commands                                  |
| expand        | expand(1)        | expand TAB characters to SPACE characters                  |
| expr          | expr(1V)         | evaluate arguments as an expression                        |
| false         | true(1)          | provide truth values                                       |
| fdformat      | fdformat(1)      | format diskettes for use under SunOS                       |
| fg            | cs(1)            | C shell built-in commands                                  |
| fgrep         | grep(1V)         | search a file for a string or regular expression           |
| file          | file(1)          | determine the type of a file by examining its contents     |
| find          | find(1)          | find files by name, or by other characteristics            |
| finger        | finger(1)        | display information about users                            |
| fmt           | fmt(1)           | simple text and mail-message formatters                    |
| fmt_mail      | fmt(1)           | simple text and mail-message formatters                    |
| fold          | fold(1)          | fold long lines for display                                |
| fontedit      | fontedit(1)      | a vfont screen-font editor                                 |
| foption       | foption(1)       | determine available floating-point code generation options |
| foreach       | cs(1)            | C shell built-in commands                                  |
| from          | from(1)          | display the sender and date of newly-arrived mail messages |
| ftp           | ftp(1C)          | file transfer program                                      |
| gcore         | gcore(1)         | get core images of running processes                       |
| get           | get(1)           | get a version of an SCCS file                              |
| get_selection | get_selection(1) | copy contents of a selection to the standard output        |
| getopt        | getopt(1)        | parse command options in shell scripts                     |
| getoptcv      | getopts(1)       | parse command options in shell scripts                     |
| getopts       | getopts(1)       | parse command options in shell scripts                     |
| gfxtool       | gfxtool(1)       | run graphics programs in a SunView window                  |
| gigipplot     | plot(1G)         | graphics filters for various plotters                      |
| glob          | cs(1)            | C shell built-in commands                                  |
| goto          | cs(1)            | C shell built-in commands                                  |
| gprof         | gprof(1)         | display call-graph profile data                            |
| graph         | graph(1G)        | draw a graph                                               |
| grep          | grep(1V)         | search a file for a string or regular expression           |
| groups        | groups(1)        | display a user's group memberships                         |
| hashstat      | cs(1)            | C shell built-in commands                                  |
| head          | head(1)          | display first few lines of specified files                 |
| help          | help(1)          | ask for help regarding SCCS errors or warnings             |
| help_viewer   | help_viewer(1)   | SunView help application                                   |
| history       | cs(1)            | C shell built-in commands                                  |
| hostid        | hostid(1)        | print the numeric identifier of the current host           |

|                       |                        |                                                               |
|-----------------------|------------------------|---------------------------------------------------------------|
| hostname              | hostname(1)            | set or print name of current host system                      |
| hplot                 | plot(1G)               | graphics filters for various plotters                         |
| i386                  | machid(1)              | return true if processor is of a given type                   |
| iAPX286               | machid(1)              | return true if processor is of a given type                   |
| iconedit              | iconedit(1)            | edit images for SunView icons, cursors and panels             |
| id                    | id(1)                  | print the user name and ID, and group name and ID             |
| if                    | csh(1)                 | C shell built-in commands                                     |
| implot                | plot(1G)               | graphics filters for various plotters                         |
| indent                | indent(1)              | indent and format a C program source file                     |
| indentpro_to_defaults | defaultsedit(1)        | create or edit default settings for SunView 1                 |
| indxbib               | indxbib(1)             | create an inverted index to a bibliographic database          |
| inline                | inline(1)              | in-line procedure call expander                               |
| input_from_defaults   | defaultsedit(1)        | create or edit default settings for SunView 1                 |
| input_from_defaults   | input_from_defaults(1) | update the mouse and keyboard from defaults                   |
| insert_brackets       | textedit_filters(1)    | filters provided with textedit(1)                             |
| install               | install(1)             | install files                                                 |
| ipcrm                 | ipcrm(1)               | remove message queue, semaphore, shared memory ID             |
| ipcs                  | ipcs(1)                | report interprocess communication facilities status           |
| jobs                  | csh(1)                 | C shell built-in commands                                     |
| join                  | join(1)                | relational database operator                                  |
| keylogin              | keylogin(1)            | decrypt and store secret key                                  |
| kill                  | kill(1)                | send a signal to a process, or terminate a process            |
| label                 | csh(1)                 | C shell built-in commands                                     |
| last                  | last(1)                | indicate last logins by user or terminal                      |
| lastcomm              | lastcomm(1)            | show the last commands executed, in reverse order             |
| ld.so                 | ld(1)                  | link editor, dynamic link editor                              |
| ld                    | ld(1)                  | link editor, dynamic link editor                              |
| ldd                   | ldd(1)                 | list dynamic dependencies                                     |
| leave                 | leave(1)               | remind you when you have to leave                             |
| lex                   | lex(1)                 | lexical analysis program generator                            |
| limit                 | csh(1)                 | C shell built-in commands                                     |
| line                  | line(1)                | read one line                                                 |
| lint                  | lint(1V)               | a C program verifier                                          |
| ln                    | ln(1)                  | make hard or symbolic links to files                          |
| load                  | load(1)                | load Sun386i clusters                                         |
| loadc                 | load(1)                | load Sun386i clusters                                         |
| lockscreen            | lockscreen(1)          | maintain SunView context and prevent unauthorized access      |
| lockscreen_default    | defaultsedit(1)        | create or edit default settings for SunView 1                 |
| lockscreen_default    | lockscreen(1)          | maintain SunView context and prevent unauthorized access      |
| logger                | logger(1)              | add entries to the system log                                 |
| login                 | login(1)               | log in to the system                                          |
| logname               | logname(1)             | get the name by which you logged in                           |
| logout                | csh(1)                 | C shell built-in commands                                     |
| look                  | look(1)                | find words in the system dictionary or lines in a sorted list |
| lookbib               | lookbib(1)             | find references in a bibliographic database                   |
| lorder                | lorder(1)              | find an ordering relation for an object library               |
| lpq                   | lpq(1)                 | display the queue of printer jobs                             |
| lpr                   | lpr(1)                 | send a job to the printer                                     |
| lprm                  | lprm(1)                | remove jobs from the printer queue                            |
| lptest                | lptest(1)              | generate lineprinter ripple pattern                           |
| ls                    | ls(1V)                 | list the contents of a directory                              |
| m4                    | m4(1V)                 | macro language processor                                      |
| m68k                  | machid(1)              | return true if processor is of a given type                   |



|                           |                        |                                                             |
|---------------------------|------------------------|-------------------------------------------------------------|
| <b>mach</b>               | <b>mach(1)</b>         | display the processor type of the current host              |
| <b>machid</b>             | <b>machid(1)</b>       | return true if processor is of a given type                 |
| <b>mail</b>               | <b>mail(1)</b>         | read or send mail messages                                  |
| <b>mailrc_to_defaults</b> | <b>defaultsedit(1)</b> | create or edit default settings for SunView 1               |
| <b>mailtool</b>           | <b>mailtool(1)</b>     | SunView interface for the mail program                      |
| <b>make</b>               | <b>make(1)</b>         | maintain, update, and regenerate related programs and files |
| <b>man</b>                | <b>man(1)</b>          | display reference manual pages; find pages by keyword       |
| <b>mesg</b>               | <b>mesg(1)</b>         | permit or deny messages on the terminal                     |
| <b>mkdir</b>              | <b>mkdir(1)</b>        | make a directory                                            |
| <b>mkstr</b>              | <b>mkstr(1)</b>        | create an error message file by massaging C source files    |
| <b>more</b>               | <b>more(1)</b>         | browse or page through a text file                          |
| <b>mt</b>                 | <b>mt(1)</b>           | magnetic tape control                                       |
| <b>mv</b>                 | <b>mv(1)</b>           | move or rename files                                        |
| <b>neqn</b>               | <b>eqn(1)</b>          | typeset mathematics                                         |
| <b>newgrp</b>             | <b>newgrp(1)</b>       | log in to a new group                                       |
| <b>nice</b>               | <b>nice(1)</b>         | run a command at low priority                               |
| <b>nl</b>                 | <b>nl(1)</b>           | line numbering filter                                       |
| <b>nm</b>                 | <b>nm(1)</b>           | print name list                                             |
| <b>nohup</b>              | <b>nohup(1V)</b>       | run a command immune to hangups and quits                   |
| <b>notify</b>             | <b>csh(1)</b>          | C shell built-in commands                                   |
| <b>nroff</b>              | <b>nroff(1)</b>        | format documents for display or line-printer                |
| <b>objdump</b>            | <b>objdump(1)</b>      | dump selected parts of a COFF object file                   |
| <b>od</b>                 | <b>od(1V)</b>          | octal, decimal, hex, and ascii dump                         |
| <b>oldccat</b>            | <b>oldcompact(1)</b>   | compress and uncompress files, and cat them                 |
| <b>oldcompact</b>         | <b>oldcompact(1)</b>   | compress and uncompress files, and cat them                 |
| <b>oldeyacc</b>           | <b>oldeyacc(1)</b>     | modified yacc allowing much improved error recovery         |
| <b>oldfilemerge</b>       | <b>oldfilemerge(1)</b> | window-based file comparison and merging program            |
| <b>oldmake</b>            | <b>oldmake(1)</b>      | maintain, update, and regenerate groups of programs         |
| <b>oldprmail</b>          | <b>oldprmail(1)</b>    | display waiting mail                                        |
| <b>oldpti</b>             | <b>oldpti(1)</b>       | phototypesetter interpreter                                 |
| <b>oldsetkeys</b>         | <b>oldsetkeys(1)</b>   | modify interpretation of the keyboard                       |
| <b>oldsun3cvt</b>         | <b>oldsun3cvt(1)</b>   | convert Sun-2 executables for use on a Sun-3                |
| <b>oldsyslog</b>          | <b>oldsyslog(1)</b>    | make a system log entry                                     |
| <b>oldtektool</b>         | <b>oldtektool(1)</b>   | SunView Tektronix 4014 terminal-emulator window             |
| <b>olduncompact</b>       | <b>oldcompact(1)</b>   | compress and uncompress files, and cat them                 |
| <b>oldvc</b>              | <b>oldvc(1)</b>        | version control                                             |
| <b>on</b>                 | <b>on(1C)</b>          | execute on remote system with local environment             |
| <b>onintr</b>             | <b>csh(1)</b>          | C shell built-in commands                                   |
| <b>organizer</b>          | <b>organizer(1)</b>    | file and directory manager                                  |
| <b>overview</b>           | <b>overview(1)</b>     | run a program from SunView that takes over the screen       |
| <b>pack</b>               | <b>pack(1)</b>         | compress and expand files                                   |
| <b>page</b>               | <b>more(1)</b>         | browse or page through a text file                          |
| <b>pagesize</b>           | <b>pagesize(1)</b>     | display the size of a page of memory                        |
| <b>passwd</b>             | <b>passwd(1)</b>       | change password file information                            |
| <b>paste</b>              | <b>paste(1)</b>        | join lines of several files                                 |
| <b>pcat</b>               | <b>pack(1)</b>         | compress and expand files                                   |
| <b>pdp11</b>              | <b>machid(1)</b>       | return true if processor is of a given type                 |
| <b>perfmeter</b>          | <b>perfmeter(1)</b>    | display system performance values in a meter or strip chart |
| <b>pg</b>                 | <b>pg(1V)</b>          | page through a file on a soft-copy terminal                 |
| <b>plot</b>               | <b>plot(1G)</b>        | graphics filters for various plotters                       |
| <b>popd</b>               | <b>csh(1)</b>          | C shell built-in commands                                   |
| <b>pr</b>                 | <b>pr(1V)</b>          | prepare file(s) for printing, perhaps in multiple columns   |
| <b>printenv</b>           | <b>printenv(1)</b>     | display environment variables currently set                 |

|                       |                            |                                                              |
|-----------------------|----------------------------|--------------------------------------------------------------|
| <b>prof</b>           | <b>prof(1)</b>             | display profile data                                         |
| <b>prs</b>            | <b>prs(1)</b>              | display selected portions an SCCS history                    |
| <b>prt</b>            | <b>prt(1)</b>              | display the delta and commentary history of an SCCS file     |
| <b>ps</b>             | <b>ps(1)</b>               | display the status of current processes                      |
| <b>ptx</b>            | <b>ptx(1)</b>              | generate a permuted index                                    |
| <b>pushd</b>          | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>pwd</b>            | <b>pwd(1)</b>              | display the pathname of the current working directory        |
| <b>quota</b>          | <b>quota(1)</b>            | display a user's disk quota and usage                        |
| <b>ranlib</b>         | <b>ranlib(1)</b>           | convert archives to random libraries                         |
| <b>rasfilter8to1</b>  | <b>rasfilter8to1(1)</b>    | convert an 8-bit deep rasterfile to a 1-bit deep rasterfile  |
| <b>rastrepl</b>       | <b>rastrepl(1)</b>         | magnify a raster image by a factor of two                    |
| <b>rcp</b>            | <b>rcp(1C)</b>             | remote file copy                                             |
| <b>rdist</b>          | <b>rdist(1)</b>            | remote file distribution program                             |
| <b>refer</b>          | <b>refer(1)</b>            | expand and insert references from a bibliographic database   |
| <b>rehash</b>         | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>repeat</b>         | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>reset</b>          | <b>tset(1)</b>             | establish or restore terminal characteristics                |
| <b>rev</b>            | <b>rev(1)</b>              | reverse the order of characters in each line                 |
| <b>rlogin</b>         | <b>rlogin(1C)</b>          | remote login                                                 |
| <b>rm</b>             | <b>rm(1)</b>               | remove (unlink) files or directories                         |
| <b>rm del</b>         | <b>rm del(1)</b>           | remove a delta from an SCCS file                             |
| <b>rmdir</b>          | <b>rm(1)</b>               | remove (unlink) files or directories                         |
| <b>roffbib</b>        | <b>roffbib(1)</b>          | format and print a bibliographic database                    |
| <b>rpcgen</b>         | <b>rpcgen(1)</b>           | an RPC protocol compiler                                     |
| <b>rsh</b>            | <b>rsh(1C)</b>             | remote shell                                                 |
| <b>rup</b>            | <b>rup(1C)</b>             | show host status of local machines (RPC version)             |
| <b>runtime</b>        | <b>runtime(1C)</b>         | show host status of local machines                           |
| <b>rusers</b>         | <b>rusers(1C)</b>          | who's logged in on local machines (RPC version)              |
| <b>rwall</b>          | <b>rwall(1C)</b>           | write to all users over a network                            |
| <b>rwho</b>           | <b>rwho(1C)</b>            | who's logged in on local machines                            |
| <b>sact</b>           | <b>sact(1)</b>             | print current SCCS file editing activity                     |
| <b>sccs</b>           | <b>sccs(1)</b>             | front end for the Source Code Control System (SCCS)          |
| <b>sccsdiff</b>       | <b>sccsdiff(1)</b>         | compare two versions of an SCCS file                         |
| <b>screenblank</b>    | <b>screenblank(1)</b>      | turn off the screen when the mouse and keyboard are idle     |
| <b>screendump</b>     | <b>screendump(1)</b>       | dump a frame-buffer image to a file                          |
| <b>screenload</b>     | <b>screenload(1)</b>       | load a frame-buffer image from a file                        |
| <b>script</b>         | <b>script(1)</b>           | make typescript of a terminal session                        |
| <b>scrolldefaults</b> | <b>defaultsedit(1)</b>     | create or edit default settings for SunView 1                |
| <b>sdiff</b>          | <b>sdiff(1)</b>            | contrast two text files by displaying them side-by-side      |
| <b>sed</b>            | <b>sed(1V)</b>             | stream editor                                                |
| <b>selection_svc</b>  | <b>selection_svc(1)</b>    | SunView selection service                                    |
| <b>set</b>            | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>setenv</b>         | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>sh</b>             | <b>sh(1)</b>               | the standard UNIX system shell                               |
| <b>shelltool</b>      | <b>shelltool(1)</b>        | run a shell (or command) in a SunView terminal window        |
| <b>shift</b>          | <b>csh(1)</b>              | C shell built-in commands                                    |
| <b>shift_lines</b>    | <b>textedit_filters(1)</b> | filters provided with <b>textedit(1)</b>                     |
| <b>size</b>           | <b>size(1)</b>             | display the size of an object file                           |
| <b>sleep</b>          | <b>sleep(1)</b>            | suspend execution for a specified interval                   |
| <b>snap</b>           | <b>snap(1)</b>             | SunView application for system and network administration    |
| <b>soelim</b>         | <b>soelim(1)</b>           | resolve and eliminate .so requests from nroff or troff input |
| <b>sort</b>           | <b>sort(1V)</b>            | sort and collate lines                                       |
| <b>sortbib</b>        | <b>sortbib(1)</b>          | sort a bibliographic database                                |



|                    |                       |                                                            |
|--------------------|-----------------------|------------------------------------------------------------|
| source             | csch(1)               | C shell built-in commands                                  |
| sparc              | machid(1)             | return true if processor is of a given type                |
| spell              | spell(1)              | report spelling errors                                     |
| spellin            | spell(1)              | report spelling errors                                     |
| spellout           | spell(1)              | report spelling errors                                     |
| spline             | spline(1G)            | interpolate smooth curve                                   |
| split              | split(1)              | split a file into pieces                                   |
| stop               | csch(1)               | C shell built-in commands                                  |
| strings            | strings(1)            | find printable strings in an object file or binary         |
| strip              | strip(1)              | remove symbols and relocation bits from an object file     |
| stty               | stty(1V)              | set or alter the options for a terminal                    |
| stty_from_defaults | defaultsedit(1)       | create or edit default settings for SunView 1              |
| stty_from_defaults | stty_from_defaults(1) | set terminal editing characters from the defaults database |
| su                 | su(1)                 | super-user, temporarily switch to a new user ID            |
| sum                | sum(1V)               | calculate a checksum for a file                            |
| sun                | machid(1)             | return true if processor is of a given type                |
| sunview            | sunview(1)            | the SunView window environment                             |
| suspend            | csch(1)               | C shell built-in commands                                  |
| swin               | swin(1)               | set or get SunView user input options                      |
| switch             | csch(1)               | C shell built-in commands                                  |
| switcher           | switcher(1)           | switch between multiple SunView desktops                   |
| symorder           | symorder(1)           | rearrange a list of symbols                                |
| sync               | sync(1)               | update the super block; force changed blocks to the disk   |
| sysex              | sysex(1)              | invoke the system exerciser                                |
| syswait            | syswait(1)            | execute a command, suspending termination until user input |
| t300               | plot(1G)              | graphics filters for various plotters                      |
| t300s              | plot(1G)              | graphics filters for various plotters                      |
| t4013              | plot(1G)              | graphics filters for various plotters                      |
| t450               | plot(1G)              | graphics filters for various plotters                      |
| tabs               | tabs(1V)              | set tab stops on a terminal                                |
| tail               | tail(1)               | display the last part of a file                            |
| talk               | talk(1)               | talk to another user                                       |
| tar                | tar(1)                | create tape archives, and add or extract files             |
| tbl                | tbl(1)                | format tables for nroff or troff                           |
| tcov               | tcov(1)               | copy a magnetic tape                                       |
| tee                | tee(1)                | construct test coverage analysis and statement profile     |
| tek                | plot(1G)              | replicate the standard output                              |
| telnet             | telnet(1C)            | graphics filters for various plotters                      |
| test               | test(1V)              | interface to remote system using TELNET protocol           |
| textedit           | textedit(1)           | return true or false according to a conditional expression |
| textedit_filters   | textedit_filters(1)   | SunView window- and mouse-based text editor                |
| tftp               | tftp(1C)              | filters provided with textedit(1)                          |
| then               | csch(1)               | trivial file transfer program                              |
| time               | time(1V)              | C shell built-in commands                                  |
| tip                | tip(1C)               | time a command                                             |
| toolplaces         | toolplaces(1)         | terminal emulator, telephone connection to a remote system |
| touch              | touch(1V)             | display SunView window locations                           |
| tput               | tput(1V)              | update the access and modification times of a file         |
| tr                 | tr(1V)                | initialize a terminal or query the terminfo database       |
| trace              | trace(1)              | translate characters                                       |
| traffic            | traffic(1C)           | trace system calls and signals                             |
| troff              | troff(1)              | SunView program to display Ethernet traffic                |
|                    |                       | typeset or format documents                                |

|            |              |                                                          |
|------------|--------------|----------------------------------------------------------|
| true       | true(1)      | provide truth values                                     |
| tset       | tset(1)      | establish or restore terminal characteristics            |
| tsort      | tsort(1)     | topological sort                                         |
| tty        | tty(1)       | display the name of the terminal                         |
| u3b15      | machid(1)    | return true if processor is of a given type              |
| u3b2       | machid(1)    | return true if processor is of a given type              |
| u3b5       | machid(1)    | return true if processor is of a given type              |
| u3b        | machid(1)    | return true if processor is of a given type              |
| ul         | ul(1)        | do underlining                                           |
| umask      | cs(1)        | C shell built-in commands                                |
| unalias    | cs(1)        | C shell built-in commands                                |
| uname      | uname(1V)    | display the name of the current system                   |
| uncompress | compress(1)  | compress or expand files, display expanded contents      |
| unexpand   | expand(1)    | expand TAB characters to SPACE characters                |
| unget      | unget(1)     | undo a previous get of an SCCS file                      |
| unhash     | cs(1)        | C shell built-in commands                                |
| unifdef    | unifdef(1)   | resolve and remove ifdef'ed lines from cpp input         |
| uniq       | uniq(1)      | remove or report adjacent duplicate lines                |
| units      | units(1)     | conversion program                                       |
| unix2dos   | unix2dos(1)  | convert text file from SunOS format to DOS format        |
| unlimit    | cs(1)        | C shell built-in commands                                |
| unload     | unload(1)    | unload Sun386i clusters                                  |
| unloadc    | unload(1)    | unload Sun386i clusters                                  |
| unpack     | pack(1)      | compress and expand files                                |
| unset      | cs(1)        | C shell built-in commands                                |
| unsetenv   | cs(1)        | C shell built-in commands                                |
| uptime     | uptime(1)    | show how long the system has been up                     |
| users      | users(1)     | display a compact list of users logged in                |
| uucp       | uucp(1C)     | system to system copy                                    |
| uudecode   | uuencode(1C) | encode a binary file, or decode its ASCII representation |
| uuencode   | uuencode(1C) | encode a binary file, or decode its ASCII representation |
| uulog      | uucp(1C)     | system to system copy                                    |
| uuname     | uucp(1C)     | system to system copy                                    |
| uusend     | uusend(1C)   | send a file to a remote host                             |
| uustat     | uustat(1C)   | uucp status inquiry and job control                      |
| uux        | uux(1C)      | remote system command execution                          |
| vacation   | vacation(1)  | reply to mail automatically                              |
| val        | val(1)       | validate an SCCS file                                    |
| vax        | machid(1)    | return true if processor is of a given type              |
| vfontinfo  | vfontinfo(1) | inspect and print out information about fonts            |
| vgrind     | vgrind(1)    | grind nice program listings                              |
| vi         | vi(1)        | visual display editor based on ex(1)                     |
| view       | vi(1)        | visual display editor based on ex(1)                     |
| vplot      | vplot(1)     | plot graphics for a Versatec printer                     |
| vswap      | vswap(1)     | convert a foreign font file                              |
| vtroff     | vtroff(1)    | troff to a raster plotter                                |
| vwidth     | vwidth(1)    | make a troff width table for a font                      |
| w          | w(1)         | who is logged in, and what are they doing                |
| wait       | wait(1)      | wait for a process to finish                             |
| wall       | wall(1)      | write to all users logged in                             |
| wc         | wc(1)        | display a count of lines, words and characters           |
| what       | what(1)      | identify the version of files under SCCS                 |
| whatis     | whatis(1)    | display a one-line summary about a keyword               |



|                 |                    |                                                             |
|-----------------|--------------------|-------------------------------------------------------------|
| <b>whereis</b>  | <b>whereis(1)</b>  | locate binary, source, and manual page for a command        |
| <b>which</b>    | <b>which(1)</b>    | locate a command; display its pathname or alias             |
| <b>while</b>    | <b>cs(1)</b>       | C shell built-in commands                                   |
| <b>who</b>      | <b>who(1)</b>      | who is logged in on the system                              |
| <b>whoami</b>   | <b>whoami(1)</b>   | display the effective current username                      |
| <b>whois</b>    | <b>whois(1)</b>    | DARPA Internet user name directory service                  |
| <b>write</b>    | <b>write(1)</b>    | write a message to another user                             |
| <b>xargs</b>    | <b>xargs(1)</b>    | construct the arguments list(s) and execute a command       |
| <b>xget</b>     | <b>xsend(1)</b>    | send or receive secret mail                                 |
| <b>xsend</b>    | <b>xsend(1)</b>    | send or receive secret mail                                 |
| <b>xstr</b>     | <b>xstr(1)</b>     | extract strings from C programs to implement shared strings |
| <b>yacc</b>     | <b>yacc(1)</b>     | yet another compiler-compiler: parsing program generator    |
| <b>yes</b>      | <b>yes(1)</b>      | be repetitively affirmative                                 |
| <b>ypcat</b>    | <b>ypcat(1)</b>    | print values in a YP data base                              |
| <b>ypmatch</b>  | <b>ypmatch(1)</b>  | print the value of one or more keys from a YP map           |
| <b>yppasswd</b> | <b>yppasswd(1)</b> | change your network password in the Yellow Pages            |
| <b>ypwhich</b>  | <b>ypwhich(1)</b>  | which host is the YP server or map master?                  |
| <b>zcat</b>     | <b>compress(1)</b> | compress or expand files, display expanded contents         |





## NAME

**ar** – create library archives, and add or extract files

## SYNOPSIS

**ar** **d|m|p|q|r|t|x** [**clouv**] [**abi position-name**] *archive* [*member-file*...]

## SYSTEM V SYNOPSIS

**ar** [**-**] **d|m|p|q|r|t|x** [**clouvs**] [**abi position-name**] *archive* [*member-file*...]

## DESCRIPTION

**ar** maintains groups of files combined into a single archive file. An archive file comprises a set of member files and header information for each file. The archive header and the headers for the file consist of printable characters (assuming that the names of the files in the archive contain printable characters), and are in a format portable across all machines. This format is described in detail in **ar(5)**. If an archive is composed of printable files, with printable file names, the entire archive is printable.

**ar** is normally used to create and update library files used by the link editor **ld(1)**, but can be used for any similar purpose.

*archive* is the name of the archive file. *member-file* is a member file contained in the archive. If this argument is omitted, the command applies to all entries in the archive. Member names have a maximum of 15 characters, except on Sun386i systems, where they have a maximum of 14 characters, longer names are truncated.

## SYSTEM V DESCRIPTION

**ar** runs **ranlib** after modifying an archive, so that the symbol table member of the archive is kept up-to-date.

## OPTIONS

You must indicate only one of: **d**, **m**, **p**, **q**, **r**, **t**, or **x**, which may be followed by one or more **Modifiers**.

- d** Delete the named files from the archive file.
- m** Move the named files to the end of the archive.
- p** Print. If no names are given, all files in the archive are written to the standard output; if no names are given, only those files are written, and they are written in the order that they appear in the archive.
- q** Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece. If this option is used to add a member to an archive, and a member with the same name as that member already exists in the archive, the old member will not be removed; two members with the same name will exist in the archive.
- r** Replace the named files in the archive.
- t** Table of contents. If no names are given, all files in the archive are listed; if names are given, only those files are listed.
- x** Extract. If no names are given, all files in the archive are extracted into the current directory; if names are given, only those files are extracted. In neither case does **x** alter the archive file.

## Modifiers

- c** Create. Suppress the message that is produced by default when *archive* is created.
- l** Local. Place temporary files in the current working directory rather than in the default temporary directory, **/tmp**.
- o** Old date. When files are extracted with the **x** option, set the “last modified” date to the date recorded in the archive.
- u** Update. Replace only those files that have changed since they were put in the archive. Used with the **r** option.

**v** Verbose. When used with the **r**, **d**, **m**, or **q** option, give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **x**, give a file-by-file description of the extraction of archive files. When used with **t**, give a long listing of information about the files. When used with **p**, write each member's name to the standard output before writing the member to the standard output.

**a** *position-name*

Place new files after *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

**b** *position-name*

Place new files before *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

**i** *position-name*

Identical to the **b** modifier.

## SYSTEM V OPTIONS

The options may be preceded by '-'.  
 Modifiers

**s** Force the regeneration of the archive symbol table even if **ar** is not invoked with a command that will modify the archive contents.

## EXAMPLES

Creating a new archive:

```
example% ar rcv archive file.o
a - file.o
```

Adding to an archive:

```
example% ar rav file.o archive next.c
a - next.c
```

Extracting from an archive:

```
example% ar xv archive file.o
x - file.o
example% ls file.o
file.o
```

Seeing the table of contents:

```
example% ar t archive
file.o
next.c
```

## FILES

/tmp/v\*. temporary files  
 /tmp

## SEE ALSO

ld(1), lorder(1), ranlib(1), ar(5)

## BUGS

If the same file is mentioned twice in an argument list, it is put in the archive twice.

The "last-modified" date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the super-user.



## NAME

**arch** – display the architecture of the current host

## SYNOPSIS

**arch**  
**arch -k**  
**arch archname**

## DESCRIPTION

**arch** displays the application architecture of the current host system.

Sun systems can be broadly classified by their *architectures*, which define what executables will run on which machines. A distinction can be made between *kernel architecture* and *application architecture* (or, commonly, just “architecture”). Machines which run different kernels due to underlying hardware differences may yet be able to run the same application programs. For example, the MC68020-based Sun-3/160 system and the MC68030-based Sun-3/80 system run different SunOS kernels, but can execute the same applications. One could describe these machines as having “the same application architecture” but “different kernel architectures.”

Executing **arch** will display the application architecture of the machine, such as **sun3**, **sun4**, etc. All machines of the same application architecture will execute the same application programs, or user binaries.

## Current Architectures

| Application architecture | Kernel architecture | Current Sun System Models                                  |
|--------------------------|---------------------|------------------------------------------------------------|
| sun2                     | sun2                | 2/50, 2/120                                                |
| sun3                     | sun3                | 3/50, 3/60, 3/75, 3/110, 3/140, 3/160, 3/180, 3/260, 3/280 |
| sun3                     | sun3x               | 3/80, 3/460, 3/470, 3/480                                  |
| sun4                     | sun4                | 4/110, 4/260, 4/280, SPARCsystem 330                       |
| sun386                   | sun386              | 386i/150, 386i/250                                         |

## OPTIONS

- k** Display the kernel architecture, such as **Sun3**, **Sun3x**, etc. This defines which specific SunOS kernel will run on the machine, and has implications only for programs that depend on the kernel explicitly (for example, **ps(1)**, **vmstat(1)**, etc.).
- archname** Return “true” (exit status 0) if user binaries for *archname* can run on the current host system, otherwise, return “false” (exit status 1). This is the preferred method for installation scripts to determine the environment of the host machine; that is, which architecture of a multi-architecture release to install on this machine. *archname* must be a valid application architecture.

## SEE ALSO

**mach(1)**, **machid(1)**

*Installing the SunOS 4.0.3*

## NAME

**cpp** – the C language preprocessor

## SYNOPSIS

```
/usr/lib/cpp [ -BCHMpPRT ] [ -undef ] [ -Dname ] [ -Dname=def ] [ -Idirectory ] [ -Uname ]
[ -Ydirectory ] [ input-file [ output-file ] ]
```

## DESCRIPTION

**cpp** is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc(1V)** command, and may also be used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc(1V)**, or another compilation command. For general-purpose macro-processing, see **m4(1V)**, and the *Programming Utilities and Libraries*.

**cpp** optionally accepts two filenames as arguments. *input-file* and *output-file* are, respectively, the input and output files for the preprocessor. They default to the standard input and the standard output.

## OPTIONS

- B** Support the C++ comment indicator `'//'`. With this indicator everything on the line after the `//` is treated as a comment.
- C** Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.
- H** Print the pathnames of included files, one per line on the standard error.
- M** Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.
- p** Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.
- P** Preprocess the input without producing the line control information used by the next pass of the C compiler.
- R** Allow recursive macros.
- T** Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.
- undef** Remove initial definitions for all predefined symbols.
- Dname** Define *name* as 1 (one). This is the same as if a **-Dname=1** option appeared on the **cpp** command line, or as if a
 

```
#define name 1
```

 line appeared in the source file that **cpp** is processing.
- Dname=def** Define *name* as if by a **#define** directive. This is the same as if a
 

```
#define name def
```

 line appeared in the source file that **cpp** is processing. The **-D** option has lower precedence than the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.



- Idirectory** Insert *directory* into the search path for **#include** files with names beginning with **'/'**. *directory* is inserted ahead of the standard list of "include" directories. Thus, **#include** files with names enclosed in double-quotes (") are searched for first in the directory of the file with the **#include** line, then in directories named with **-I** options, and lastly, in directories from the standard list. For **#include** files with names enclosed in angle-brackets (<>), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.
- Uname** Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the application architecture of the system:
- |                              |                                                                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Operating System:            | <b>ibm, gcos, os, tss and unix</b>                                                                                                    |
| Hardware:                    | <b>interdata, pdp11, u370, u3b, u3b2, u3b5, u3b15, u3b20d, vax, mc68000, mc68010, mc68020, ns32000, iAPX286, i386, sparc, and sun</b> |
| UNIX system variant:         | <b>RES, and RT</b>                                                                                                                    |
| The <b>lint(1V)</b> command: | <b>lint</b>                                                                                                                           |
- The symbols **sun** and **unix** are defined for all Sun systems, as is the value returned by the **mach** command. For Sun-4 systems, this value would be **sparc**. In addition, **mc68000** is defined for Sun-2, Sun-3, and Sun-3x systems.
- Ydirectory** Use directory *directory* in place of the standard list of directories when searching for **#include** files.

## USAGE

### Directives

All **cpp** directives start with a hash symbol (**#**) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial **#** for proper indentation.

**#define name token-string**

Replace subsequent instances of *name* with *token-string*.

**#define name(argument [, argument] ... ) token-string**

There can be no space between *name* and the **'('**. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *argument* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

**#undef name**

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

**#include "filename"**

**#include <filename>**

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the **<filename>** notation is used, *filename* is only searched for in the standard "include" directories. See the **-I** and **-Y** options above for more detail. No additional tokens are permitted on the directive line after the final **"'"** or **>'**.

**#line *integer-constant* "*filename*"**

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If "*filename*" is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

**#if *constant-expression***

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including '&&', '||', and ',', are legal in *constant-expression*. The '?:' operator, and the unary '-', '!', and '~' operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: '**defined** (*name*)' or '**defined** *name*'. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **sizeof** operator is not available.

**#ifdef *name***

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a **-D** option, and in the absence of an intervening **#undef** directive. No additional tokens are permitted on the directive line after *name*.

**#ifndef *name***

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name*.

**#elif *constant-expression***

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive was defined.
- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

**#else** This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

**#endif** End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. Each such directive must have a matching **#endif**.

**Macros**

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a) | \a|
abc(xyz)
```



is the seven characters “|‘xyz|’” (SPACE, vertical-bar, backquote, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc xyz
printf("abc");
```

does not expand `abc` in the second line, since it is inside a quoted string that is not part of a `#define` macro definition.

Macros are not expanded while processing a `#define` or `#undef`. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces `abc`. The token appearing immediately after an `#ifdef` or `#ifndef` is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces “`#define hello goodbye hello`”.

#### Output

Output consists of a copy of the input file, with modifications, plus lines of the form:

```
#lineno " filename " "level"
```

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=1), the first such line after an include file has been exited (*level*=2), or any other such line (*level* is empty).

#### Details

##### Directory Search Order

`#include` files is:

1. The directory of the file that contains the `#include` request (that is, `#include` is relative to the file being scanned when the request is made).
2. The directories specified by `-I` options, in left-to-right order.
3. The standard directory(s) (`/usr/include` on SunOS systems).

##### Special Names

Two special names are understood by `cpp`. The name `__LINE__` is defined as the current line number (a decimal integer) as known by `cpp`, and `__FILE__` is defined as the current filename (a C string) as known by `cpp`. They can be used anywhere (including in macros) just as any other defined name.

##### Newline Characters

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a `#define` statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

##### Comments

Comments are removed (unless the `-C` option is used on the command line). Comments are also ignored, except that a comment terminates a token.

**FILES**

**/usr/include**                standard directory for **#include** files

**SEE ALSO**

**cc(1V)**, **m4(1V)**

*Programming Utilities and Libraries*

**DIAGNOSTICS**

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

**NOTES**

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

**#include <file.h>**

should be used, rather than one with an absolute path, like:

**#include "/usr/include/file.h"**

**cpp** warns about the use of the absolute pathname.



**NAME**

**crontab** – install, edit, remove or list a user's crontab file

**SYNOPSIS**

```
crontab [ filename ]
crontab -e [ username ]
crontab -l [ username ]
crontab -r [ username ]
```

**DESCRIPTION**

**crontab** copies the specified file, or the standard input if no file is specified, into a directory that holds all users' **crontab** files. A user's **crontab** file lists commands that are to be executed on behalf of that user at specified times on specified dates; the format of these files is described in **crontab(5)**.

If the file **/var/spool/cron/at.allow** exists, only users whose username appears in it can use **crontab**. If that file does *not* exist, however, **crontab** checks the **/var/spool/cron/at.deny** file to determine if the user should be denied the use of **crontab**. If neither file exists, only the super-user is allowed to submit a **crontab** job. If **at.allow** does not exist and **at.deny** exists and is empty, global usage is permitted. The **allow/deny** files consist of one user name per line.

**OPTIONS**

- e Make a copy of the current user's **crontab** file, or create an empty file if it does not exist, and edit that file. The **vi(1)** editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor. When editing is complete, install the file as the user's **crontab** file if it was modified. If a *username* is given, the specified user's **crontab** file is edited, rather than the current user's **crontab** file; this may only be done by the super-user.
- l List the user's **crontab** file.
- r Remove the current user's **crontab** file from the **crontab** directory. If a *username* is given, the specified user's **crontab** file is removed, rather than the current user's **crontab** file; this may only be done by the super-user.

**FILES**

|                                 |                       |
|---------------------------------|-----------------------|
| <b>/var/spool/cron</b>          | main cron directory   |
| <b>/var/spool/cron/crontabs</b> | spool area            |
| <b>/var/spool/cron/at.allow</b> | list of allowed users |
| <b>/var/spool/cron/at.deny</b>  | list of denied users  |

**SEE ALSO**

**sh(1)**, **crontab(5)**, **cron(8)**

**WARNINGS**

If you inadvertently enter the **crontab** command with no argument, do not attempt to get out by typing **CTRL-D**. This removes all entries in your **crontab** file. Instead, exit by typing your interrupt character (normally **CTRL-C**).





## NAME

ed – basic line editor

## SYNOPSIS

ed [ - ] [ -sx ] [ -p *string* ] [ *filename* ]

## DESCRIPTION

ed is the most basic line editor of the UNIX system. Although superseded by ex(1) and vi(1) for most purposes, ed is still used by various system utilities.

ed operates on a copy of *filename*, called a buffer, and overwrites a file only when you issue the w (write) command. ed provides line oriented editing commands to display or change lines, to insert and delete lines from the buffer, to move or copy lines within the buffer, or to substitute character strings within lines.

## OPTIONS

- 
- s        Suppress printing of character counts (by e, r, and w commands), diagnostics (by e and q commands), and the ! prompt (after a ! command). Also, suppress printing the ? diagnostic before overwriting unsaved changes in the buffer.
- x        Edit an encrypted file (see crypt(1) for details).
- p *string*    Use *string* as the editing prompt in command mode.

## USAGE

## Command Structure

ed commands have a simple and regular structure. They consist of an optional *address*, or two optional *addresses* separated by a comma or semicolon, then a single-character *command*, which may be followed by a *parameter* for that *command*:

[*address* [ , *address* ] ] *command* [ *parameter* ]

If only one *address* is specified, operations are performed on that line. If two *addresses* are specified, ed performs the operation on the inclusive range of lines. Commands that requires an *address* use certain addresses by default, typically the address of the current line.

For example, 1,10p means “print (display) lines 1 through 10” (two addresses), 5a means “append text after line 5” (one address), and d means “delete the current line” (no address with the current line used as default). The meaning of *parameter* varies for each operation — for the move (m) and transfer (t) operations, for instance, it is the line that the addressed lines are to be moved to or transferred after. For reading (r) and writing (w) a file, *parameter* specifies the name of the file that is to be read or written.

ed is extremely terse in its interaction with the user. Its normal response to most problems is simply a question mark (?). This may happen when ed cannot find a specified line in the buffer, or if a search for a regular expression fails in a substitute (s) command. The h command prints a somewhat more complete diagnostic for the most recent error encountered; the H command requests that the diagnostic be printed for all errors.

## Addresses

Lines can be addressed in several ways:

- nnn*      By line number. Lines in the buffer are numbered relative to the start of the buffer. When displayed, line numbers are not physically present with the text of the file or buffer.
- \$        The last line of the buffer.
- .
- The current line. ed keeps track of the line on which you last performed an operation. This line is called the *current line*. You can address this line by typing a “dot” character.
- ±*n*      By relative line number. Address the line number that is *n* lines higher, or *n* lines lower than the current line.
- '*c*      Address the line marked with the mark character *c*, which must be a lower-case letter. Lines are marked with the k command, described below.



**/RE/** An *RE* is a Regular Expression, described under **Regular Expressions** below. When enclosed by slashes, *RE* addresses the first line found by searching for a matching string. The search proceeds forward from the line following the current line, and wraps through the beginning of the buffer to include all preceding lines, as well as the current line.

**?RE?** An *RE* enclosed in question marks addresses the first line containing a match found by searching backward from the line preceding the current line. The search wraps through the end of the buffer to include all lines following the current line (in reverse order), as well as the current line.

#### **address±n**

An address followed by a plus sign (+) or a minus sign (−), followed by a decimal number, specifies that address plus or minus the indicated number of lines. (The plus sign may be omitted.) If the address is omitted, the current line is used as the base. For example, '31−3' addresses line 28 in the buffer.

#### **address±**

If an address ends with '+' or '−', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the previous rule, the address '−' refers to the line preceding the current line. (To maintain compatibility with earlier versions of *ed*, the character '^' is equivalent to '−'.) Trailing '+' and '−' characters have a cumulative effect, so '−−' refers to the current line, less 2.

, By itself, a comma stands for the address pair '1,\$'.

; A semicolon by itself stands for the pair '.,\$'.

By default for a given command. If you do not specify an address for a command to operate on, a command that requires an address supplies one by default. This is typically the current line.

A pair of addresses separated by a comma signifies an inclusive range of lines, and the current line is not changed unless the command changes it. When addresses are separated by a semicolon, however, the current line is set to the address preceding the semicolon before any subsequent addresses are interpreted. This feature can be used to determine the starting line for forward and backward searches using '/', and '?'. The second address of any two-address sequence must correspond to a line that occurs later in the buffer than that of the first address.

### **Regular Expressions**

*ed* supports a limited form of regular-expression notation, which can be used in a line address to specify lines by content. A regular expression (RE) specifies a set of character strings to match against — such as "any string containing digits 5 through 9" or "only lines containing uppercase letters." A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or *patterns* are used to address lines in the buffer (see **Addresses**, above), and also for selecting strings to be replaced using the *s* (substitute) command.

Where multiple matches are present in a line, a regular expression matches the the *longest* of the *leftmost* matching strings.

Regular expressions can be built up from the following "single-character" RE's:

**c** Any ordinary character not listed below. An ordinary character matches itself.

**\** Backslash. When followed by a special character, the RE matches the "quoted" character. A backslash followed by one of <, >, (, ), {, or }, represents an *operator* in a regular expression, as described below.

**.** Dot. Matches any single character except NEWLINE.

**^** As the leftmost character, a caret (or circumflex) constrains the RE to match the leftmost portion of a line. A match of this type is called an "anchored match" because it is "anchored" to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than the start of the RE.



- \$ As the rightmost character, a dollar sign constrains the RE to match the rightmost portion of a line. The \$ character loses its special meaning if it appears in any position other than at the end of the RE.
- ^RE\$ The construction ^RE\$ constrains the RE to match the entire line.
- \< The sequence \< in an RE constrains the one-character RE immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.
- \> The sequence \> in an RE constrains the one-character RE immediately following it only to match something at the end of a "word."
- [c...] A nonempty string of characters, enclosed in square brackets matches any single character in the string. For example, [abcxyz] matches any single character from the set 'abcxyz'. When the first character of the string is a caret (^), then the RE matches any character *except* NEWLINE and those in the remainder of the string. For example, '[^45678]' matches any character except '45678'. A caret in any other position is interpreted as an ordinary character.
- [!c...] The right square bracket does not terminate the enclosed string if it is the first character (after an initial '^', if any), in the bracketed string. In this position it is treated as an ordinary character.
- [l-r] The minus sign, between two characters, indicates a range of consecutive ASCII characters to match. For example, the range '[0-9]' is equivalent to the string '[0123456789]'. Such a bracketed string of characters is known as a *character class*. The '-' is treated as an ordinary character if it occurs first (or first after an initial '^') or last in the string.
- d Delimiter character. The character used to delimit an RE within a command is special for that command (for example, see how / is used in the g command, below).

The following rules and special characters allow for constructing RE's from single-character RE's:

- A concatenation of RE's matches a concatenation of text strings, each of which is a match for a successive RE in the search pattern.
- \* A single-character RE, followed by an asterisk (\*) matches *zero* or more occurrences of the single-character RE. Such a pattern is called a *closure*. For example, [a-z][a-z]\* matches any string of one or more lower case letters.
- \{m\}
- \{m,\}
- \{m,n\} A one-character RE followed by \{m\}, \{m,\}, or \{m,n\} is an RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be nonnegative integers less than 256; \{m\} matches *exactly* *m* occurrences; \{m,\} matches *at least* *m* occurrences; \{m,n\} matches *any number* of occurrences *between* *m* and *n*, inclusively. Whenever a choice exists, the RE matches as many occurrences as possible.
- \(...\) An RE enclosed between the character sequences \(( and\) matches whatever the unadorned RE matches, but saves the string matched by the enclosed RE in a numbered substring register. There can be up to nine such substrings in an RE, and parenthesis operators can be nested.
- \n Match the contents of the *n*th substring register from the current RE. This provides a mechanism for extracting matched substrings. For example, the expression ^(\.\*)\1\$ matches a line consisting entirely of two adjacent non-null appearances of the same string. When nested parenthesized substrings are present, *n* is determined by counting occurrences of \(( starting from the left.
- // The null RE (/) is equivalent to the last RE encountered.

#### Commands

The commands *a* for *append*, *c* for *change*, and *i* for *insert*, allow you to add new text to the buffer. While accepting new text, ed is said to be in *input mode*. While in input mode, *no* commands are recognized; all character input is inserted into the buffer. To exit from input mode, enter a dot (.) on a line by itself; ed then reverts to command mode. Or, you can interrupt ed (typically with CTRL-C), in which case it displays



a ? and returns to command mode.

Commands may accept zero, one, or two addresses. Commands that accept no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when too few addresses are given; if more addresses are given than such a command requires, only the last ones are used.

In the following list of ed commands, the default addresses are shown in parentheses; the parenthesized addresses are *not* part of the command.

It is generally illegal for more than one command to appear on a line. However, any command (except e, f, r, or w) may be followed by l, n, or p in which case the current line is either listed, numbered or printed, respectively.

(.) **a**

*text*

. Append *text*. Add lines of *text* into the buffer after the addressed line. The resulting current line is the last line of input, or the addressed line if no text is entered. Address 0 is legal for this command, in which case the *text* is placed at the beginning of the buffer. The maximum number of characters per input line (from a terminal) is 256, including the final NEWLINE.

(.) **c**

*text*

. Change lines. Delete the addressed lines, and then accept lines of *text* to replace them. c accepts one or two addresses; the default is the current line. The resulting current line is the last line of input, or the line preceding the deleted lines if no text is entered.

(...) **d** Delete the addressed lines from the buffer. d accepts one or two addresses; the default is the current line. The resulting current line is the line following the last one deleted; if the deleted lines were at the end of the buffer, the new last line is the resulting current line.

**e filename**

Edit a file. Delete the entire contents of the buffer, and then read in the named file. The resulting current line is the last line of the buffer. e reports the number of characters read into the buffer, and sets *filename* to be the current file (for use as a default filename in subsequent commands). If no *filename* is given, the current filename, if any, is used (see the f command, below). If *filename* is replaced by a shell (sh(1)) command prefaced with a '!', the shell command is executed and its output is read into the buffer after the current line. Such a shell command is *not* used as the current filename. e displays a ? if the buffer has not been written out since the last change made — another e command in response to the ? forces the command to take effect.

**E filename**

The E command is like e, except that the editor does not check for changes to the buffer since the last w command was performed.

**f filename**

Display or set the current filename. If *filename* is given as an argument, the file (f) command changes the current filename to *filename*; otherwise, it prints the current filename.

(1,\$) **g/RE/command-list**

The global (g) command performs *command-list* on all lines in the range of addresses that match *RE*. ed executes *command-list* for each matching line in succession, setting the current line to each in turn. *command-list* can contain a single command, or it can be continued across input lines, with one ed command per line, by escaping all but the last NEWLINE with a \ character. Operations that place ed into input mode (a, i, and c), are permitted in *command-list*; the final '.' terminating text input may be omitted if it is the last line of the *command-list*. g, G, v, and V commands, however, are *not* permitted. An empty *command-list* is equivalent to the p command.



**(1,\$) G/RE/**

The interactive **G** (Global) command, selects all lines that match the given *RE*. Then, each selected line is made current, and any *one* command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) can be performed upon that line. A NEWLINE acts as a null command; an **&** reexecutes the most recent command. Commands entered during execution of the **G** command can address and affect lines other than the current line. The **G** command can be terminated by an interrupt (typically CTRL-D).

**h** Help. Display a short error message that explains the reason for the most recent **?** diagnostic.

**H** Automatic printing of help diagnostics. Toggle between printing the **?** diagnostic, or automatically printing diagnostic messages as well.

**(.)i**

*text*

**.** Insert Text. Insert the given *text* into the buffer, above the addressed line. **i** accepts one *address*; the default is the current line. The resulting current line is the last line of input; if no text is input, it is the line just before the addressed line. This command differs from the **a** command only in the placement of the input text; Address 0 is not allowed for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

**(...+1)j** Join Lines. Remove the NEWLINE character from between the two addressed lines. The defaults are the current line and the line following. If exactly one address is given, this command does nothing. The joined line is the resulting current line.

**(.)kc** Mark the addressed line with the name *c*, a lower-case letter. The address-form, '*c*', addresses the line marked by *c*. **k** accepts one *address*; the default is the current line. The current line is left unchanged.

**(... )l** List nonprinting characters. Print the addressed lines in an unambiguous way: a few nonprinting characters, such as TAB and BACKSPACE are represented by visually mnemonic overstrikes. All other nonprinting characters are shown in octal, with long lines folded. **l** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

**(... )maddress**

Move addressed lines to just after *address*. Address 0 is legal, and moves the addressed line(s) to the beginning of the file. An error results if *address* falls within the range of lines to move. **m** accepts two addresses to specify a range of lines to move; the default is the current line. The resulting current line is the last of the moved lines.

**(... )n** Number the displayed lines. Print the addressed lines, preceding each with its line number and a TAB character. **n** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **n** command can be appended to any command other than **e**, **f**, **r**, or **w**.

**(... )p** Print the addressed lines. **p** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

**P** Toggle prompting on or off. When prompting is in effect, the editor prompts with a **\*** for commands. A subsequent **P** command turns prompting off.

**q** Quit. Exit from **ed**. Note, however, that the buffer is *not* automatically written out; you must write any changes to be saved with the **w** command; **ed** warns you once if you have not saved your changes (unless the **'-**' option is in effect). A second **q** forces **ed** to exit regardless, destroying the buffer's contents.

**Q** Force quit. This is the same as **q**, but you do not get any warning if you have not previously written out the buffer. **ed** simply exits.



(**\$**) **r** *filename*

Read in the contents of *filename*, after the addressed line. If *filename* is not given, the current filename, if any, is used (see the **e** and **f** commands). The current filename is *not* altered; if there is no current filename, *filename* becomes the current filename. **r** accepts one *address*; the default is **\$**. Address 0 is legal for **r**, in which case the file is read in at the beginning of the buffer. If the read is successful, the number of characters read is typed. The resulting current line is the last line read in from the file. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a **!**, the shell command is executed and its output is read in. Such a shell command is *not* remembered as the current filename.

(**...**) **s**/*RE*/*rs*/

(**...**) **s**/*RE*/*rs*/**g**

(**...**) **s**/*RE*/*rs*/**n**

Substitute. Search each addressed line for the first occurrence of a string matching the specified *RE*, and replace it with *rs*, the replacement string. If **g** (global suffix) is appended to the command, replace *all* (non-overlapped) matching strings in each addressed line with the replacement string *rs*. Note: the *g* suffix is distinct from the *g* command. If a number *n* is appended, replace only the *n*'th occurrence of the matched string on each addressed line. **s** accepts one or two addresses; the default is the current line. The resulting current line is the last line on which a substitution is made. An error results if *RE* matches no strings in the addressed line or range. Any character (other than SPACE or NEWLINE) can be used instead of **/** to delimit *RE* and *rs*. As with *RE*'s in addresses, you can refer to the entire string matched by *RE* with an **'&'**; you can refer to parenthesized substrings within *RE* using **\1... \n**. When **%** is the only character in *rs*, the *rs* from the most recent substitute command is used as the current *rs*. The **%** loses its special meaning when it is in a replacement string of more than one character, or if it is preceded by a backslash.

A line may be split by substituting a NEWLINE character into it. The NEWLINE in the *replacement* must be escaped by preceding with an **'\'**. Such substitutions cannot be done as part of a **g** or **v** command list.

(**...**) **t***address*

Transfer. Transpose a copy of the addressed range of lines to just after the given *address*. **t** (transfer) is like **m** (move), except that it copies of the lines, rather than moving them. **t** accepts two addresses preceding the operation letter, the current address is the default. The resulting current line is the last line copied. Address 0 is allowed.

**u** Undo. Reverse the effect of the most recent command that modified the buffer. A second **u** undoes the undo operation.

(**1,\$**) **v**/*RE*/*command-list*

This command is the same as the global command **g** except that the *command-list* is executed with **'.'** initially set to every line that does *not* match the *RE*.

(**1,\$**) **V**/*RE*

Similar to the **G** command, except that the lines selected are those that do *not* match the *RE*.

(**1,\$**) **w** *filename*

Write the addressed lines to *filename*. If the file does not exist, **ed** creates it. The current filename is *not* altered; if there is no current filename, then *filename* becomes current. If no *filename* is given, the current filename, if any, is used. **w** accepts one or two addresses; the default is all lines in the file. The current line is unchanged. If the command is successful, the number of characters written is displayed. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a **'!'**, the shell command is executed with standard input taken from the addressed lines. Such a shell command is *not* remembered as the current filename.

(**1,\$**) **W** *filename*

Like **w**, but append the addressed lines onto the named file.

**x** Encrypt the file. **ed** prompts for an encryption key from the standard input. Subsequent **e**, **r**, and



w commands encrypt and decrypt the text with this key (see `crypt(1)`). An empty key turns off encryption. Encryption can also be specified on the command line with the `-x` option.

**(*\$*) =** Display the line number of the addressed line; the current line remains unchanged.

#### **!shell-command**

Run a shell command. *shell-command* is a (Bourne shell) command line. `ed` replaces the unescaped character `%` with the current filename; if a `!` appears as the first character of the shell command, it is replaced with the text of the previous shell command. (`!!` repeats the last shell command.) If any such expansion is performed, the expanded line is echoed. The current line is unchanged.

#### **address**

#### **NEWLINE**

An address, alone on a line, prints the addressed line. A NEWLINE alone is equivalent to `+.1p`, which is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, `ed` prints a `?` and returns to *its* command level.

#### **File Format Specification Support**

`ed` supports the `fspec(5)` formatting capability for displaying lines. When the first line of a file is a format specification of the form:

```
<:ts[,ts]...smax:>
```

where *ts* is the column number of a tab stop and *smax* is the maximum line length for display purposes, and with the terminal in `'stty -tabs'` or `'stty tab3'` mode (see `stty(1V)` for details), the indicated tab stops are used in displayed lines. While inserting text, however, tab stops are set to every eighth column.

#### **ENVIRONMENT**

**TMPDIR** If this environment variable is set and is not null, its value is used in place of `/usr/tmp` as the directory in which the temporary file is placed.

#### **FILES**

|                           |                                                |
|---------------------------|------------------------------------------------|
| <code>/usr/tmp/ed#</code> | temporary; # is the process number             |
| <code>ed.hup</code>       | file for saved work if the terminal is hung up |

#### **SEE ALSO**

`crypt(1)`, `ex(1)`, `grep(1V)`, `sed(1V)`, `sh(1)`, `stty(1V)`, `vi(1)`, `regex(3)`, `fspec(5)`

*Editing Text Files*

#### **LIMITATIONS**

The following limitations apply:

- 512 characters per line.
- 256 characters per global command-list.
- 1024 characters per filename.
- The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, `ed` discards ASCII NUL characters and all characters after the last NEWLINE. Files (such as executable images) that contain characters not in the ASCII set (bit 8 on) cannot be edited using `ed`.

If a file is not terminated by a NEWLINE character, `ed` adds one and prints a message saying that it has done so.

If the closing delimiter of an RE or of a replacement string (such as `/`) would be the last character before a NEWLINE, that delimiter can be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

|                      |                        |
|----------------------|------------------------|
| <code>s/s1/s2</code> | <code>s/s1/s2/p</code> |
| <code>g/s1</code>    | <code>g/s1/p</code>    |
| <code>?s1</code>     | <code>?s1?</code>      |

**DIAGNOSTICS**

**?** For command errors.

**?file:error**

For an inaccessible file (use the **h** (help) and **H** (Help) commands for detailed explanations).

If changes have been made in the buffer since the last **w** command, **ed** issues a warning **?** when a command is given that would destroy the buffers contents. A second **e** or **q** command at this point will take effect. The **'-'** and **-s** command-line options inhibit this feature.

**CAVEATS AND BUGS**

A **!** command cannot be subject to a **g** or a **v** command.

The sequence **\n** in an RE does not match a NEWLINE character.

Files encrypted directly with the **crypt(1)** command with the null key cannot be edited.

The encryption facilities of **ed** are not available on software shipped outside the U.S.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file, the editor exits at the first failure of a command in that file.





**NAME**

eject – eject floppy disk from an autoeject floppy drive

**SYNOPSIS**

**eject** [ **-d** | **-f** | **-n** | *device* | *nickname* ]

**AVAILABILITY**

Sun-3/80 systems only.

**DESCRIPTION**

**eject** is used for those removable media devices that do not have a manual eject button. The device may be specified by its name or by a nickname; if no device is specified the default device is used.

Only devices that support **eject** under program control respond to this command.

**eject** can also display its default device and a list of nicknames.

**OPTIONS**

**-d** Display the name of the default device to be ejected.

**-f** Force the device to eject even if it is busy.

**-n** Display the nickname to device name translation table.

*device* Specify which device to eject, by the name it appears in the directory **/dev**.

*nickname*

Specify which device to eject, by its nickname as known to this command.

**FILES**

**/dev/rfd0a**

**SEE ALSO**

**fd(4S)**

**DIAGNOSTICS**

A short help message is printed if an unknown flag is specified. A diagnostic is printed if the device name cannot be opened or does not support **eject**.

**BUGS**

There ought to be a way to change the default on a per-user basis.



**NAME**

**env** – obtain or alter environment variables for command execution

**SYNOPSIS**

**env** [ - ] [ *name=value ...* ] [ *command* ]

**DESCRIPTION**

**env** obtains the current **environment**, modifies it according to its arguments, and executes the command with the modified environment that results.

If no *command* is specified, the resulting environment is displayed.

**OPTIONS**

|                   |                                                                                                                                                                      |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| –                 | Ignore the environment that would otherwise be inherited from the current shell.<br>Restricts the environment for <i>command</i> to that specified by the arguments. |
| <i>name=value</i> | Set the environment variable <i>filename</i> to <i>value</i> and add it to the environment.                                                                          |

**SEE ALSO**

**sh(1), execve(2), profil(2), environ(5V)**





## NAME

**expr** – evaluate arguments as a logical, arithmetic, or string expression

## SYNOPSIS

**expr** *argument...*

## DESCRIPTION

**expr** evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

**expr** \| *expr*

Return the first *expr* if it is neither NULL nor 0, otherwise returns the second *expr*.

**expr** \|& *expr*

Return the first *expr* if neither *expr* is NULL or 0, otherwise returns 0.

**expr** { =, \>, \>=, \<, \<=, != } *expr*

Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

**expr** { +, - } *expr*

Addition or subtraction of integer-valued arguments.

**expr** { \\*, /, % } *expr*

Multiplication, division, or remainder of the integer-valued arguments.

**string** : *regular-expression*

**match** *string regular-expression*

The two forms of the matching operator above are synonymous. The matching operators : and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are “anchored” (treated as if they begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (0 on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

**substr** *string integer-1 integer-2*

Extract the substring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

**index** *string character-list*

Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

**length** *string*

Return the length (that is, the number of characters) of *string*.

( **expr** ) Parentheses may be used for grouping.

## SYSTEM V DESCRIPTION

The operators **substr**, **index**, and **length** are not supported.

## EXAMPLES

1. **a='expr \$a + 1'**  
Adds 1 to the shell variable a.
2. **# 'For \$a equal to either "/usr/abc/file" or just "file"'**  
**expr \$a : '.\*\/(.\*)' \ \$a**  
Returns the last segment of a path name (that is, the filename part). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).
3. **# A better representation of example 2.**  
**expr // \$a : '.\*\/(.\*)'**  
The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.
4. **expr \$VAR : '.\*'**  
Returns the number of characters in \$VAR.

## SEE ALSO

ed(1), sh(1), test(1V)

## EXIT CODE

**expr** returns the following exit codes:

- |   |                                         |
|---|-----------------------------------------|
| 0 | if the expression is neither null nor 0 |
| 1 | if the expression is null or 0          |
| 2 | for invalid expressions.                |

## DIAGNOSTICS

- |                             |                                             |
|-----------------------------|---------------------------------------------|
| <b>syntax error</b>         | for operator/operand errors                 |
| <b>non-numeric argument</b> | if arithmetic is attempted on such a string |
| <b>division by zero</b>     | if an attempt to divide by zero is made     |

## BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If \$a is an =, the command:

**expr \$a = '='**

looks like:

**expr = = =**

as the arguments are passed to **expr** (and they will all be taken as the = operator). The following works:

**expr X\$a = X=**

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

**example% expr index expurgatorious length**

**syntax error**

**example%**

generates the 'syntax error' message as shown instead of the value 1 as you might expect.





**NAME**

**fdformat** – format diskettes for use under SunOS

**SYNOPSIS — Sun386i SYSTEMS**

**fdformat** [ **-L** ] [ **-2** ]

**fdformat** [ **-efLV** ] [ *device* ]

**AVAILABILITY**

Sun386i and Sun-3/80 systems only.

**DESCRIPTION**

**fdformat** is a program for formatting diskettes to use with the SunOS operating system. All new blank diskettes must be formatted before use. **fdformat** formats and verifies each track on the diskette, and terminates if it finds any bad sectors. **fdformat** destroys all existing data on the diskette.

By default, **fdformat** formats a 1.44 megabyte high density diskette. Use the **-L** or **-l** option to format low density diskettes.

On Sun386i systems, use the **-2** option to format diskettes in the optional external 5 1/4" floppy drive.

On Sun-3/80 systems the default device is the internal floppy drive, **/dev/rfd0c**.

To format a diskette for use under MS-DOS, use the MS-DOS format command in a DOS window on the Sun386i system.

**OPTIONS**

- e** Eject the diskette when done. (Sun-3/80 systems only.)
- f** Force. Do not ask for confirmation before starting format. (Sun-3/80 systems only.)
- l** Format a low density diskette (720 kilobyte) diskette. (Sun-3/80 systems only.)
- L** Format a low density diskette (720 kilobyte) diskette.
- v** Verify the floppy diskette after formatting. (Sun-3/80 systems only.)
- 2** Format a diskette in the optional external 5 1/4" floppy drive. (Sun386i systems only.)

**FILES — Sun386i SYSTEMS**

**/dev/rfd0c**  
**/dev/rfd10c**  
**/dev/rfd2c**  
**/dev/rfd12c**

**SEE ALSO**

**dos(1)**, **fd(4s)**

**BUGS**

The SunOS system currently does not support bad sector mapping on diskettes. Therefore, a diskette is unusable if **fdformat** finds an error (bad sector).



**NAME**

**file** – determine the type of a file by examining its contents

**SYNOPSIS**

**file** [ **-f** *ffile* ] [ **-cL** ] [ **-m** *mfile* ] *filename*...

**DESCRIPTION**

**file** performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, **file** examines the first 512 bytes and tries to guess its language.

**file** uses the file */etc/magic* to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

**OPTIONS**

- c** Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **-c**.
- f** *ffile* Get a list of filenames to identify from *ffile*.
- L** If a file is a symbolic link, test the file the link references rather than the link itself.
- m** *mfile* Use *mfile* as the name of an alternate magic number file.

**EXAMPLE**

This example illustrates the use of **file** on all the files in a specific user's directory:

```
example% pwd
/usr/blort/misc
example% file *
code:                mc68020 demand paged executable
code.c:              c program text
counts:              ascii text
doc:                 roff, nroff , or eqn input text
empty.file:          empty
libz:                 archive random library
memos:               directory
project:             symbolic link to /usr/project
script:              executable shell script
titles:              ascii text
s5.stuff:             cpio archive
example%
```

**FILES**

*/etc/magic*

**SEE ALSO**

**magic(5)**

**BUGS**

**file** often makes mistakes. In particular, it often suggests that command files are C programs. Does not recognize Pascal or LISP.

## NAME

**make** – maintain, update, and regenerate related programs and files

## SYNOPSIS

```
make [ -f makefile ] ... [ -d ] [ -dd ] [ -D ] [ -DD ] [ -e ] [ -i ] [ -k ] [ -n ] [ -p ] [ -P ] [ -q ] [ -r ]
      [ -s ] [ -S ] [ -t ] [ -z ] [ target ... ] [ macro=value ... ]
```

## DESCRIPTION

**make** executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called *dependencies*. Since each dependency is a target, it may have dependencies of its own. Targets, dependencies, and sub-dependencies comprise a tree structure that **make** traces when deciding whether or not to rebuild a *target*.

**make** recursively checks each *target* against its dependencies, beginning with the first target entry in *makefile* if no *target* argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing, or to be older than any of its dependencies, **make** rebuilds it. Optionally with this version of **make**, a target can be treated as out-of-date when the commands used to generate it have changed since the last time the target was built.

To build a given target, **make** executes the list of commands, called a *rule*. This rule may be listed explicitly in the target's *makefile* entry, or it may be supplied implicitly by **make**.

When no *makefile* is specified with a **-f** option:

- If there is a file named **makefile** in the working directory, **make** uses that file. If, however, there is an SCCS history file (SCCS/s.**makefile**) which is newer, **make** attempts to retrieve and use the most recent version.
- In the absence of the above file(s), if a file named **Makefile** is present in the working directory, **make** attempts to use it. If there is an SCCS history file (SCCS/s.**Makefile**) that is newer, **make** attempts to retrieve and use the most recent version.

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*.

If a *target* has no *makefile* entry, or if its entry has no rule, **make** attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. (Each method is described under USAGE below.)

- Pattern matching rules.
- Implicit rules, read in from a user-supplied *makefile*.
- Standard implicit rules (also known as suffix rules), typically read in from the file `/usr/include/make/default.mk`.
- SCCS retrieval. **make** retrieves the most recent version from the SCCS history file (if any). See the description of the `.SCCS_GET`: special-function target for details.
- The rule from the `.DEFAULT`: target entry, if there is such an entry in the *makefile*.

If there is no *makefile* entry for a *target*, if no rule can be derived for building it, and if no file by that name is present, **make** issues an error message and halts.

## OPTIONS

**-f** *makefile*

Use the description file *makefile*. A '-' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the standard set of implicit rules and predefined macros. When more than one '**-f** *makefile*' argument pair appears, **make** uses the concatenation of those files, in order of appearance.

**-d**

Display the reasons why **make** chooses to rebuild a target; **make** displays any and all dependencies that are newer. In addition, **make** displays options read in from the `MAKEFLAGS` environment variable.



- dd** Display the dependency check and processing in vast detail.
- D** Display the text of the makefiles read in.
- DD** Display the text of the makefiles, **default.mk** file, the state file, and all hidden-dependency reports.
- e** Environment variables override assignments within makefiles.
- i** Ignore error codes returned by commands. Equivalent to the special-function target **'IGNORE:'**.
- k** When a nonzero error status is returned by a rule, or when **make** cannot find a rule, abandon work on the current target, but continue with other dependency branches that do not depend on it.
- n** No execution mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed. However, if a command line contains a reference to the **\$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Reading Makefiles and the Environment**).
- p** Print out the complete set of macro definitions and target descriptions.
- P** Merely report dependencies, rather than building them.
- q** Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.
- r** Do not read in the default makefile, **/usr/include/make/default.mk**.
- s** Silent mode. Do not print command lines before executing them. Equivalent to the special-function target **'.SILENT:'**.
- S** Undo the effect of the **-k** option. Stop processing when a non-zero exit status is returned by a command.
- t** Touch the target files (bringing them up to date) rather than performing their rules. *This can be dangerous when files are maintained by more than one person.* When the **.KEEP\_STATE:** target appears in the makefile, this option updates the state file just as if the rules had been performed.
- z** Propagate **-d** and **-D** to nested (recursive) commands through the **MAKEFLAGS** macro.

**macro=value**

Macro definition. This definition overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments.

## USAGE

Refer to *Doing More with SunOS: Beginner's Guide* and **make** in *Programming Utilities and Libraries* for tutorial information about **make**.

### Reading Makefiles and the Environment

When **make** first starts, it reads the **MAKEFLAGS** environment variable to obtain any the following options specified present in its value: **-e**, **-i**, **-k**, **-l**, **-n**, **-p**, **-q**, **-r**, **-s**, **-S**, **-t**, or **-z**. (Within the **MAKEFLAGS** value, the leading **'—'** character for the option string is omitted.) **make** then reads the command line for additional options, which also take effect.

Next, **make** reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for retrieving SCCS files. If present, **make** uses the file **default.mk** in the current directory; otherwise it reads the file **/usr/include/make/default.mk**, which contains the standard definitions and rules.

Use the directive:

```
include /usr/include/make/default.mk
```

in your local **default.mk** file to include them.



Next, **make** imports variables from the environment (unless the **-e** option is in effect), and treats them as defined macros. Because **make** uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When **-e** is in effect, however, environment variables are read in *after* all makefiles have been read. In that case, the environment variables take precedence over definitions in the makefile.

Next, **make** reads the state file, **.make.state** in the local directory if it exists, and then any makefiles you specify with **-f**, or one of **makefile** or **Makefile** as described above.

Next, (after reading the environment if **-e** is in effect), **make** reads in any macro definitions supplied as command line arguments. These override macro definitions in the makefile and the environment both, but only for the **make** command itself.

**make** exports environment variables, using the most recently defined value. Macro definitions supplied on the command line are not normally exported, unless the macro is also an environment variable.

**make** does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, **make** exports its value as defined on the command line. Unless **-e** is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros **MAKEFLAGS**, **MAKE**, **KEEP\_STATE**, **SHELL**, **HOST\_ARCH**, **TARGET\_ARCH**, **HOST\_MACH**, and **TARGET\_MACH** are special cases. See **Special-Purpose Macros** below, for details.

#### Makefile Target Entries

A target entry has the following format:

```
target... [:|::] [dependency] ... [; command] ...
      [command]
...
```

The first line contains the name of a target, or a space-separated list of target names, terminated with a colon or double colon. If a list of targets is given, this is equivalent to having a separate entry of the same form for each target. The colon(s) may be followed by a *dependency*, or a dependency list. **make** checks this list before building the target. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a single Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise the rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or # begins another target or macro definition.

#### Makefile Special Characters

##### Global

**#** Start a comment. The comment ends at the next NEWLINE. If the # follows the TAB in a command line, that line is passed to the shell (which also treats # as the start of a comment).

##### **include** *filename*

If the word **include** appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. **include** files can be nested to a depth of no more than about 16. If *filename* is a macro reference, it is expanded.



*Targets and Dependencies*

- : Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.
- :: Target terminator for alternate dependencies. When used in place of a ':' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in the first alternate, it is built according to the rule for that entry. When out-of-date with respect to dependencies in another alternate, it is built according to the rule in that other entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.
- target* [+ *target* ... ] : Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per **make** run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.
- % Pattern matching wild card metacharacter. Like the \* shell wild card, % matches any string of zero or more characters in a target name or dependency, in the target portion of a conditional macro definition, or within a pattern replacement macro reference. Note: only one % can appear in a target, dependency-name, or pattern-replacement macro reference.

*Macros*

- = Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading and trailing white space characters are stripped from the value. A word break following the = is implied.
- \$ Macro reference. The following character, or the parenthesized or bracketed string, is interpreted as a macro reference: **make** expands the reference (including the \$) by replacing it with the macro's value.
- ()
- { } Macro-reference name delimiters. A parenthesized or bracketed word appended to a \$ is taken as the name of the macro being referred to. Without the delimiters, **make** recognizes only the first character as the macro name.
- \$\$ A reference to the dollar-sign macro, the value of which is the character '\$'. Used to pass variable expressions beginning with \$ to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.
- \\$ Escaped dollar-sign character. Interpreted as a literal dollar sign within a rule.
- += When used in place of '=', appends a string to a macro definition (must be surrounded by white space, unlike '=').
- := Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

*Rules*

- **make** ignores any nonzero error code returned by a command line for which the first non-TAB character is a '-'. This character is not passed to the shell as part of the command line. **make** normally terminates when a command returns nonzero status, unless the -i or -k options, or the .IGNORE: special-function target is in effect.
- @ If the first non-TAB character is a @, **make** does not print the command line before executing it. This character is not passed to the shell.
- ? Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

**!** Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the '?' dynamic macro (for example, '\$?').

When any combination of '-', '@', '?', or '!' appear as the first characters after the TAB, all that are present apply. None are passed to the shell.

#### Special-Function Targets

When incorporated in a makefile, the following target names perform special-functions:

##### **.DEFAULT:**

If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to retrieve a current version. **make** ignores any dependencies for this target.

**.DONE:** If defined in the makefile, **make** processes this target and its dependencies after all other targets are built. This target is also performed when **make** halts with an error, unless the **.FAILED** target is defined.

##### **.FAILED:**

This target, along with its dependencies, is performed instead of **.DONE** when defined in the makefile and **make** halts with an error.

##### **.IGNORE:**

Ignore errors. When this target appears in the makefile, **make** ignores non-zero error codes returned from commands.

**.INIT:** If defined in the makefile, this target and its dependencies are built before any other targets are processed.

##### **.KEEP\_STATE:**

If this target appears in the makefile, **make** updates the state file, **.make.state**, in the current directory. This target also activates command dependencies, and hidden dependency checks.

##### **.MAKE\_VERSION:**

A target-entry of the form:

**.MAKE\_VERSION: VERSION-number**

enables version checking. If the version of **make** differs from the version indicated, **make** issues a warning message.

##### **.PRECIOUS:**

List of files not to delete. **make** does not remove any of the files listed as dependencies for this target when interrupted. **make** normally removes the current target when it receives an interrupt.

##### **.SCCS\_GET:**

This target contains the rule for retrieving the current version of an SCCS file from its history file. To suppress automatic retrieval, add an entry for this target with an empty rule to your makefile.

##### **.SILENT:**

Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

##### **.SUFFIXES:**

The suffixes list for selecting implicit rules (see **The Suffixes List**).



*Clearing Special Targets*

In this version of **make**, you can clear the definition of the following special targets by supplying entries for them with no dependencies and no rule:

**.DEFAULT**, **.DONE**, **.FAILED**, **.INIT**, **.SCCS\_GET**, and **.SUFFIXES**

*Command Dependencies*

When the **.KEEP\_STATE**: target appears in the makefile, **make** checks the command for building a target against the state file, **.make.state**. If the command has changed since the last **make** run, **make** rebuilds the target.

*Hidden Dependencies*

When the **.KEEP\_STATE**: target appears in the makefile, **make** reads reports from **cpp(1)** and other compilation processors for any "hidden" files, such as **#include** files. If the target is out of date with respect to any of these files, **make** rebuilds it.

*Macros*

Entries of the form

*macro=value*

define macros. *macro* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. **make** strips both leading and trailing white space in accepting the value.

Subsequent references to the macro, of the forms: **\$(name)** or **\${name}** are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro references can contain references to other macros, in which case nested references are expanded first.

*Suffix Replacement Macro References*

Substitutions within macros can be made as follows:

**\$(name:string1=string2)**

where *string1* is either a suffix, or a word to be replaced in the macro definition, and *string2* is the replacement suffix or word. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

*Pattern Replacement Macro References*

Pattern matching replacements can also be applied to macros, with a reference of the form:

**\$(name:op%os=np%ns)**

where *op* is the existing (old) prefix and *os* is the existing (old) suffix, *np* and *ns* are the new prefix and new suffix, respectively, and the pattern matched by **%** (a string of zero or more characters), is carried forward from the value being replaced. For example:

```
PROGRAM=fabricate
DEBUG=$(PROGRAM:%=tmp/%-g)
```

sets the value of **DEBUG** to **tmp/fabricate-g**.

Note: pattern replacement macro references cannot be used in the dependency line of a pattern matching rule; the **%** characters are not evaluated independently.

*Appending to a Macro*

Words can be appended to macro values as follows:

*macro += word ...*

*Special-Purpose Macros*

When the **MAKEFLAGS** variable is present in the environment, **make** takes options from it, in combination with options entered on the command line. **make** retains this combined value as the **MAKEFLAGS** macro, and exports it automatically to each command or shell it invokes.

Note: flags passed by way of **MAKEFLAGS** are only displayed when the **-d**, or **-dd** options are in effect.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **-n** option for any line in which it is referred to. This allows nested invocations of **make** written as:

**\$(MAKE) ...**

to run recursively, with the **-n** flag in effect for all commands but **make**. This lets you use '**make -n**' to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a **'-'**. **MFLAGS** is not exported automatically.

The **SHELL** macro, when set to a single-word value such as **/usr/bin/csh**, indicates the name of an alternate shell to use. The default is **/bin/sh**. Note: **make** executes commands that contain no shell metacharacters itself. Built-in commands, such as **dirs** in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of **-e**. To be sure it is set properly, you must define this macro within every makefile that requires it.

The **KEEP\_STATE** environment variable has the same effect as the **.KEEP\_STATE**: special-function target. It enables command dependencies, hidden dependencies and writing of the state file.

The following macros are provided for use with cross-compilation:

#### **HOST\_ARCH**

The machine architecture of the host system. By default, this is the output of the **arch(1)** command prepended with **'—'**. Under normal circumstances, this value should never be altered by the user.

#### **TARGET\_ARCH**

The machine architecture of the target system. By default, the output of **arch**, prepended with **'—'**.

#### **HOST\_MACH**

The machine architecture of the host system. By default, this is the output of the **mach(1)**, prepended with **'—'**. Under normal circumstances, this value should never be altered by the user.

#### **TARGET\_MACH**

The machine architecture of the target system. By default, the output of **mach**, prepended with **'—'**.

#### **Dynamic Macros**

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; if you were to define them, **make** would simply override the definition.

- \$\*** The basename of the current target, derived as if selected for use with an implicit rule. In the case of pattern matching rules, the value is the string matched by the **'%'**.
- \$<** The name of a dependency file, derived as if selected for use with an implicit rule.
- \$@** The name of the current target.
- \$?** The list of dependencies that are newer than the target. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a **'?'**. See the description of **'?'**, under **Makefile Special Tokens**, above. You can force this check with the **!** command-line prefix.
- %** The name of the library member being processed. (See **Library Maintenance**, below.)



To refer to a dynamic macro within a dependency list, precede the reference with an additional '\$' character (for example, '\$\$@'). Because **make** assigns \$< and \$\* as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

These macros can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case F or D, respectively (and enclosing the resulting name in parentheses or braces). Thus, '\$(@D)' refers to the directory part of the string '\$@'; if there is no directory part, '.' is assigned. '\$(@F)' refers to the filename part.

#### Conditional Macro Definitions

A macro definition of the form:

*target-list := macro = value*

indicates that when processing any of the targets listed *and their dependencies*, *macro* is to be set to the *value* supplied. Note that if a conditional macro is referred to in a dependency list, the \$ must be delayed (use \$\$ instead). Also, *target-list* may contain a % pattern, in which case the macro will be conditionally defined for all targets encountered that match the pattern. A pattern replacement reference can be used within the *value*.

You can temporarily append to a macro's value with a conditional definition of the form:

*target-list := macro += value*

#### Predefined Macros

**make** supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any of these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each is expected to take. If -r is in effect, **make** does not supply these macro definitions.

| Table of Predefined Macros   |                                                                      |                                                                                                                                                                                                                 |
|------------------------------|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use                          | Macro                                                                | Default Value                                                                                                                                                                                                   |
| Library Archives             | AR<br>ARFLAGS                                                        | ar<br>rv                                                                                                                                                                                                        |
| Assembler Commands           | AS<br>ASFLAGS<br>COMPILE.s<br>COMPILE.S                              | as<br>\$(AS) \$(ASFLAGS) \$(TARGET_MACH)<br>\$(CC) \$(ASFLAGS) \$(CPPFLAGS) \$(TARGET_MACH) -c                                                                                                                  |
| C Compiler Commands          | CC<br>CFLAGS<br>CPPFLAGS<br>COMPILE.c<br>LINK.c                      | cc<br>\$(CC) \$(CFLAGS) \$(CPPFLAGS) \$(TARGET_ARCH) -c<br>\$(CC) \$(CFLAGS) \$(CPPFLAGS) \$(LDFLAGS) \$(TARGET_ARCH)                                                                                           |
| FORTRAN 77 Compiler Commands | FC<br>FFLAGS<br>COMPILE.f<br>LINK.f<br>COMPILE.F<br>LINK.F           | f77<br>\$(FC) \$(FFLAGS) \$(TARGET_ARCH) -c<br>\$(FC) \$(FFLAGS) \$(TARGET_ARCH) \$(LDFLAGS)<br>\$(FC) \$(FFLAGS) \$(CPPFLAGS) \$(TARGET_ARCH) -c<br>\$(FC) \$(FFLAGS) \$(CPPFLAGS) \$(LDFLAGS) \$(TARGET_ARCH) |
| Link Editor Command          | LD<br>LDFLAGS                                                        | ld                                                                                                                                                                                                              |
| lex Command                  | LEX<br>LFLAGS<br>LEX.l                                               | lex<br>\$(LEX) \$(LFLAGS) -t                                                                                                                                                                                    |
| lint Command                 | LINT<br>LINTFLAGS<br>LINT.c                                          | lint<br>\$(LINT) \$(LINTFLAGS) \$(CPPFLAGS) \$(TARGET_ARCH)                                                                                                                                                     |
| Modula 2 Commands            | M2C<br>M2FLAGS<br>MODFLAGS<br>DEFFLAGS<br>COMPILE.def<br>COMPILE.mod | m2c<br>\$(M2C) \$(M2FLAGS) \$(DEFFLAGS) \$(TARGET_ARCH)<br>\$(M2C) \$(M2FLAGS) \$(MODFLAGS) \$(TARGET_ARCH)                                                                                                     |
| Pascal Compiler Commands     | PC<br>PFLAGS<br>COMPILE.p<br>LINK.p                                  | pc<br>\$(PC) \$(PFLAGS) \$(CPPFLAGS) \$(TARGET_ARCH) -c<br>\$(PC) \$(PFLAGS) \$(CPPFLAGS) \$(LDFLAGS) \$(TARGET_ARCH)                                                                                           |
| Ratfor Compilation Commands  | RFLAGS<br>COMPILE.r<br>LINK.r                                        | \$(FC) \$(FFLAGS) \$(RFLAGS) \$(TARGET_ARCH) -c<br>\$(FC) \$(FFLAGS) \$(RFLAGS) \$(TARGET_ARCH) \$(LDFLAGS)                                                                                                     |
| rm Command                   | RM                                                                   | rm -f                                                                                                                                                                                                           |
| sccs Command                 | SCCSFLAGS<br>SCCSGETFLAGS                                            | -s                                                                                                                                                                                                              |
| yacc Command                 | YACC<br>YFLAGS<br>YACC.y                                             | yacc<br>\$(YACC) \$(YFLAGS)                                                                                                                                                                                     |
| Suffixes List                | SUFFIXES                                                             | .o .c .c~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l<br>.I~ .mod .mod~ .sym .def .def~ .p .p~ .r .r~<br>.y .y~ .h .h~ .sh .sh~ .cps .cps~                                                                               |



**Implicit Rules**

When a target has no entry in the makefile, **make** attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; the corresponding dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, **make** allows you to define your own, either by pattern, or by suffix.

**Pattern Matching Rules**

A target entry of the form:

```
tp%ts: dp%ds
      rule
```

is a pattern matching rule, in which *tp* is a target prefix, *ts* is a target suffix, *dp* is a dependency prefix, and *ds* is a dependency suffix (any of which may be null). The % stands for a basename of zero or more characters that is matched in the target, and is used to construct the name of a dependency. When **make** encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the \$@ and \$< dynamic macros as placeholders for the target and dependency names. The dynamic macro \$\* is set to the string matched by the % wild card. Other, regular dependencies may occur in the dependency list. An entry of the form:

```
tp%ts: [dependency ...] dp%ds [dependency ...]
      rule
```

is a valid pattern matching rule.

**Suffix Rules**

When no pattern matching rule applies, **make** checks the target name to see if it ends with a suffix in the known suffixes list. If so, **make** checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

```
DsTs:
      rule
```

where *Ts* is the suffix of the target, *Ds* is the suffix of the dependency file, and *rule* is the rule for building a target in the class. Both *Ds* and *Ts* must appear in the suffixes list. (A suffix need not begin with a '.' to be recognized.)

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the .c rule could be used to build an executable program named **file** from a C source file named **'file.c'**.

| <i>Table of Standard Implicit (Suffix) Rules</i> |                           |                                                                                             |
|--------------------------------------------------|---------------------------|---------------------------------------------------------------------------------------------|
| <i>Use</i>                                       | <i>Implicit Rule Name</i> | <i>Command Line</i>                                                                         |
| <i>Assembly Files</i>                            | <b>.s.o</b>               | \$(COMPILE.s) -o \$@ \$<                                                                    |
|                                                  | <b>.s.a</b>               | \$(COMPILE.s) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |
|                                                  | <b>.S.o</b>               | \$(COMPILE.S) -o \$@ \$<                                                                    |
|                                                  | <b>.S.a</b>               | \$(COMPILE.S) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |
| <i>C Files</i>                                   | <b>.c</b>                 | \$(LINK.c) -o \$@ \$< \$(LDLIBS)                                                            |
|                                                  | <b>.c.ln</b>              | \$(LINT.c) \$(OUTPUT_OPTION) -i \$<                                                         |
|                                                  | <b>.c.o</b>               | \$(COMPILE.c) \$(OUTPUT_OPTION) \$<                                                         |
|                                                  | <b>.c.a</b>               | \$(COMPILE.c) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |
| <i>FORTRAN 77 Files</i>                          | <b>.f</b>                 | \$(LINK.f) -o \$@ \$< \$(LDLIBS)                                                            |
|                                                  | <b>.f.o</b>               | \$(COMPILE.f) \$(OUTPUT_OPTION) \$<                                                         |
|                                                  | <b>.f.a</b>               | \$(COMPILE.f) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |
|                                                  | <b>.F</b>                 | \$(LINK.F) -o \$@ \$< \$(LDLIBS)                                                            |
|                                                  | <b>.F.o</b>               | \$(COMPILE.F) \$(OUTPUT_OPTION) \$<                                                         |
|                                                  | <b>.F.a</b>               | \$(COMPILE.F) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |
| <i>lex Files</i>                                 | <b>.l</b>                 | \$(RM) \$*.c<br>\$(LEX.l) \$< > \$*.c<br>\$(LINK.c) -o \$@ \$*.c \$(LDLIBS)<br>\$(RM) \$*.c |
|                                                  | <b>.l.c</b>               | \$(RM) \$@<br>\$(LEX.l) \$< > \$@                                                           |
|                                                  | <b>.l.ln</b>              | \$(RM) \$*.c<br>\$(LEX.l) \$< > \$*.c<br>\$(LINT.c) -o \$@ -i \$*.c<br>\$(RM) \$*.c         |
|                                                  | <b>.l.o</b>               | \$(RM) \$*.c<br>\$(LEX.l) \$< > \$*.c<br>\$(COMPILE.c) -o \$@ \$*.c<br>\$(RM) \$*.c         |
| <i>Modula 2 Files</i>                            | <b>.mod</b>               | \$(COMPILE.mod) -o \$@ -e \$@ \$<                                                           |
|                                                  | <b>.mod.o</b>             | \$(COMPILE.mod) -o \$@ \$<                                                                  |
|                                                  | <b>.def.sym</b>           | \$(COMPILE.def) -o \$@ \$<                                                                  |
| <i>NeWS</i>                                      | <b>.cps.h</b>             | cps \$*.cps                                                                                 |
| <i>Pascal Files</i>                              | <b>.p</b>                 | \$(LINK.p) -o \$@ \$< \$(LDLIBS)                                                            |
|                                                  | <b>.p.o</b>               | \$(COMPILE.p) \$(OUTPUT_OPTION) \$<                                                         |
| <i>Ratfor Files</i>                              | <b>.r</b>                 | \$(LINK.r) -o \$@ \$< \$(LDLIBS)                                                            |
|                                                  | <b>.r.o</b>               | \$(COMPILE.r) \$(OUTPUT_OPTION) \$<                                                         |
|                                                  | <b>.r.a</b>               | \$(COMPILE.r) -o \$% \$<<br>\$(AR) \$(ARFLAGS) \$@ \$%<br>\$(RM) \$%                        |



| <i>Table of Standard Implicit (Suffix) Rules (continued)</i> |                           |                                                                                      |
|--------------------------------------------------------------|---------------------------|--------------------------------------------------------------------------------------|
| <i>Use</i>                                                   | <i>Implicit Rule Name</i> | <i>Command Line</i>                                                                  |
| <i>SCCS Files</i>                                            | <b>.SCCS_GET</b>          | <b>sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@ -G\$@</b>                             |
| <i>Shell Scripts</i>                                         | <b>.sh</b>                | <b>cat \$&lt; &gt;\$@<br/>chmod +x \$@</b>                                           |
| <i>yacc Files</i>                                            | <b>.y</b>                 | <b>\$(YACC.y) \$&lt;<br/>\$(LINK.c) -o \$@ y.tab.c \$(LDLIBS)<br/>\$(RM) y.tab.c</b> |
|                                                              | <b>.y.c</b>               | <b>\$(YACC.y) \$&lt;<br/>mv y.tab.c \$@</b>                                          |
|                                                              | <b>.y.ln</b>              | <b>\$(YACC.y) \$&lt;<br/>\$(LINT.c) -o \$@ -i y.tab.c<br/>\$(RM) y.tab.c</b>         |
|                                                              | <b>.y.o</b>               | <b>\$(YACC.y) \$&lt;<br/>\$(COMPILE.c) -o \$@ y.tab.c<br/>\$(RM) y.tab.c</b>         |

**make** reads in the standard set of implicit rules from the file `/usr/include/make/default.mk`, unless `-r` is in effect, or there is a `default.mk` file in the local directory that does not include that file.

#### The Suffixes List

The suffixes list is given as the list of dependencies for the `‘.SUFFIXES:’` special-function target. The default list is contained in the `SUFFIXES` macro (See *Table of Predefined Macros* for the standard list of suffixes). You can define additional `.SUFFIXES:` targets; a `.SUFFIXES` target with no dependencies clears the list of suffixes. Order is significant within the list; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

**.SUFFIXES:**

**.SUFFIXES: *suffixes* \$(SUFFIXES)**

A tilde (~) indicates that if a dependency file with the indicated suffix (minus the ~) is under SCCS its most recent version should be retrieved, if necessary, before the target is processed.

#### Library Maintenance

A target name of the form:

***lib(member ...)***

refers to a member, or a space-separated list of members, in an `ar(1V)` library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

***lib(%s): %s***

where `.s` is the suffix of the member; this suffix is typically `.o` for object libraries.

A target name of the form

***lib((symbol))***

refers to the member of a randomized object library (see `ranlib(1)`) that defines the entry point named *symbol*.

#### Command Execution

Command lines are executed one at a time, *each by its own process or shell*. Shell commands, notably `cd`, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a `@`, if there is a `‘.SILENT:’` entry in the makefile, or if **make** is run with the `-s` option. Although the `-n` option specifies printing without execution, lines containing the macro `$(MAKE)` are executed regardless, and lines containing the `@` special

character are printed. The **-t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

#### Bourne Shell Constructs

To use the Bourne shell **if** control structure for branching, use a command line of the form:

```
if expression ; \
then command ; \
    ... ; \
else ; \
    ... ; \
fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one (shell) command line.

To use the Bourne shell **for** control structure for loops, use a command line of the form:

```
for var in list ; \
do command ; \
    ... ; \
done
```

To refer to a shell variable, use an escaped dollar-sign (**\\$**). This prevents expansion of the dollar-sign by **make**.

To incorporate the standard output of a shell command in a macro, use a definition of the form:

```
MACRO :sh =command
```

The command is executed only once, standard error output is discarded, and NEWLINE characters are replaced with SPACES. If the command has a non-zero exit status, **make** halts with an error.

To capture the output of a shell command in a macro reference, use a reference of the form:

```
$(MACRO :sh)
```

where **MACRO** is the name of a macro containing a valid Bourne shell command line. In this case, the command is executed whenever the reference is evaluated. As with shell command substitutions, the reference is replaced with the standard output of the command. If the command has a non-zero exit status, **make** halts with an error.

#### Signals

INT and QUIT signals received from the keyboard halt **make** and remove the target file being processed unless that target is in the dependency list for **PRECIOUS**.

#### EXAMPLES

This makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
    $(LINK.c) -o $@ a.o b.o
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

The following makefile uses implicit rules to express the same dependencies:

```
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o b.o: incl.h
```



## FILES

|                                     |                                                                                           |
|-------------------------------------|-------------------------------------------------------------------------------------------|
| <b>makefile</b>                     |                                                                                           |
| <b>Makefile</b>                     | current version(s) of <b>make</b> description file                                        |
| <b>SCCS/s.makefile</b>              |                                                                                           |
| <b>SCCS/s.Makefile</b>              | SCCS history files for the above <b>makefile</b> (s)                                      |
| <b>default.mk</b>                   | default file for user-defined targets, macros, and implicit rules                         |
| <b>/usr/include/make/default.mk</b> |                                                                                           |
|                                     | <b>makefile</b> for standard implicit rules and macros (not read if <b>default.mk</b> is) |
| <b>.make.state</b>                  | state file in the local directory                                                         |

## SEE ALSO

**ar**(1V), **cc**(1V), **cd**(1), **get**(1), **lex**(1), **ranlib**(1), **passwd**(5),  
*Doing More with SunOS: Beginner's Guide*  
*Programming Utilities and Libraries*

## DIAGNOSTICS

**make** returns an exit status of 1 when it halts as a result of an error. Otherwise it returns an exit status of 0.

**Do not know how to make target. Stop.**

There is no **makefile** entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

**\*\*\* target removed.**

**make** was interrupted while building *target*. Rather than leaving a partially-completed version that is newer than its dependencies, **make** removes the file named *target*.

**\*\*\* target not removed.**

**make** was interrupted while building *target* and *target* was not present in the directory.

**\*\*\* target could not be removed, reason**

**make** was interrupted while building *target*, which was not removed for the indicated reason.

**Read of include file 'file' failed**

The **makefile** indicated in an **include** directive was not found, or was inaccessible.

**Loop detected when expanding macro value 'macro'**

A reference to the macro being defined was found in the definition.

**Could not write state file 'file'**

You used the **.KEEP\_STATE**: target, but do not have write permission on the state file.

**\*\*\* Error code n**

The previous shell command returned a nonzero error code.

**\*\*\* signal message**

The previous shell command was aborted due to a signal. If '**- core dumped**' appears after the message, a **core** file was created.

**Conditional macro conflict encountered**

Displayed only when **-d** is in effect, this message indicates that two or more parallel targets currently being processed depend on a target which is built differently for each by virtue of conditional macros. Since the target cannot simultaneously satisfy both dependency relationships, it is conflicted.

## BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a '**-**'.

Filenames with the characters '**=**', '**:**', or '**@**', do not work.

You cannot build **file.o** from **lib(file.o)**.

Options supplied by **MAKEFLAGS** should be reported for nested **make** commands. Use the **-d** option to find out what options the nested command picks up from **MAKEFLAGS**.

This version of **make** is incompatible in certain respects with previous versions:

- The **-d** option output is much briefer in this version. **-dd** now produces the equivalent voluminous output.
- **make** attempts to derive values for the dynamic macros '\$\*', '\$<', and '\$?', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead

either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as well, even though the documentation stated otherwise.)

- **make** no longer searches the current directory for SCCS history files.
- Suffix replacement in macro references are now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of **make** will work with earlier versions.

If there is no **default.mk** file in the current directory, and the file **/usr/include/make/default.mk** is missing, **make** stops before processing any targets. To force **make** to run anyway, create an empty **default.mk** file in the current directory.

Once a dependency is made, **make** assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the **\$?** macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to compiler commands when **\$?** is used in a rule.

Pattern replacement macro references cannot be used in the dependency line of a pattern matching rule.

Unlike previous versions, this version of **make** strips a leading **./** from the value of the **'\$@'** dynamic macro.

With automatic SCCS retrieval, this version of **make** does not support tilde suffix rules.





**NAME**

**mt** – magnetic tape control

**SYNOPSIS**

**mt** [ *-f tapename* ] *command* [ *count* ]

**DESCRIPTION**

**mt** sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable **TAPE** is used. If **TAPE** does not exist, **mt** uses the device **/dev/rmt12**. *tapename* must refer to a raw (not block) tape device. By default, **mt** performs the requested operation once; multiple operations may be performed by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

**mt** returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized or if **mt** was unable to open the specified tape drive, and 2 if an operation failed.

**OPTIONS**

|                  |                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>eof, weof</b> | Write <i>count</i> EOF marks at the current position on the tape.                                                         |
| <b>fsf</b>       | Forward space <i>count</i> files. The tape is positioned on the first block of the file.                                  |
| <b>fsr</b>       | Forward space <i>count</i> records.                                                                                       |
| <b>bsf</b>       | Back space <i>count</i> files. The tape is positioned on the first block of the file.                                     |
| <b>bsr</b>       | Back space <i>count</i> records.                                                                                          |
| <b>bsfm</b>      | Back space <i>count</i> file marks. The tape is positioned on the beginning-of-tape side of the file mark.                |
| <b>asf</b>       | Absolute space to <i>count</i> file number. This is equivalent to a <b>rewind</b> followed by a <b>fsf</b> <i>count</i> . |

For the following commands, *count* is ignored:

|                         |                                                                                                                                   |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>eom</b>              | Space to the end of recorded media on the tape (SCSI only). This is useful for appending files onto previously written tapes.     |
| <b>rewind</b>           | Rewind the tape.                                                                                                                  |
| <b>offline, rewofll</b> | Rewind the tape and, if appropriate, take the drive unit offline by unloading the tape.                                           |
| <b>status</b>           | Print status information about the tape unit.                                                                                     |
| <b>retension</b>        | Rewind the tape completely, then wind it forward to the end of the reel and back to beginning-of-tape to smooth out tape tension. |
| <b>erase</b>            | Erase the entire tape.                                                                                                            |

**FILES**

|                  |                                      |
|------------------|--------------------------------------|
| <b>/dev/rmt*</b> | raw magnetic tape interface          |
| <b>/dev/rar*</b> | raw Archive cartridge tape interface |
| <b>/dev/rst*</b> | raw SCSI tape interface              |
| <b>/dev/rmt*</b> | raw Xylogics tape interface          |

**SEE ALSO**

**ar(4S)**, **mtio(4)**, **st(4S)**, **tm(4S)**, **xt(4S)** **environ(5V)**

**BUGS**

Not all devices support all options, in particular **retension** and **bsfm**. For example, **ar(4S)** currently does not support the **fsr**, **bsf**, or **bsr** options. The half-inch tape driver, **/dev/rmt\***, does not support the **retension** option.



## NAME

**mv** — move or rename files

## SYNOPSIS

```
mv [-] [-f] filename1 filename2
mv [-] [-f] directory1 directory2
mv [-] [-f] filename ... directory
```

## DESCRIPTION

**mv** moves files and directories around in the file system. A side effect of **mv** is to rename a file or directory. The three major forms of **mv** are shown in the synopsis above.

The first form of **mv** moves (changes the name of) *filename1* to *filename2*. If *filename2* already exists, it is removed before *filename1* is moved. If *filename2* has a mode which forbids writing, **mv** prints the mode (see **chmod**(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise **mv** exits.

The second form of **mv** moves (changes the name of) *directory1* to *directory2*, only if *directory2* does not already exist — if it does, the third form applies.

The third form of **mv** moves one or more *filenames* (may also be directories) with their original names, into the last *directory* in the list.

**mv** refuses to move a file or directory onto itself.

## OPTIONS

- Interpret all the following arguments to **mv** as file names. This allows file names starting with minus.
- f Force. Override any mode restrictions and the **-i** option. The **-f** option also suppresses any warning messages about modes which would potentially restrict overwriting.
- i Interactive mode. **mv** displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with **y**, **mv** moves the specified file or directory, otherwise **mv** does nothing with that file or directory.

## SEE ALSO

**cp**(1), **ln**(1), **chmod**(2), **rename**(2)

## DIAGNOSTICS

**mv: pathname: rename: Permission denied**

Attempted to move *pathname* into a directory that did not have write permission.

## BUGS

If *filename1* and *filename2* are on different file systems, then **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Modification times may be different than expected when **mv** must copy the file's data, rather than simply updating a directory entry.

**mv** will not move a directory from one file system to another. Use **cp**(1) instead.

**NAME**

intro – introduction to device drivers, protocols, and network interfaces

**DESCRIPTION**

This section describes device drivers, high-speed network interfaces, and protocols available under SunOS. The system provides drivers for a variety of hardware devices, such as disks, magnetic tapes, serial communication lines, mice and frame buffers, as well as virtual devices such as pseudo-terminals and windows. SunOS provides hardware support and a network interface for the 10-Megabit Ethernet, along with interfaces for the IP protocol family and a STREAMS-based Network Interface Tap (NIT) facility.

In addition to describing device drivers that are supported by the 4.3BSD operating system, this section contains subsections that describe:

- SunOS-specific device drivers, under '4S'.
- Protocol families, under '4F'.
- Protocols and raw interfaces, under '4P'.
- STREAMS modules, under '4M'.
- Network interfaces, under '4N'.

**Configuration**

The SunOS kernel can be configured to include or omit many of the device drivers described in this section. The CONFIG section of the manual page gives the line(s) to include in the kernel configuration file for each machine architecture on which a device is supported. If no specific architectures are indicated, the configuration syntax applies to all Sun systems.

The GENERIC kernel is the default configuration for SunOS. It contains all of the optional drivers for a given machine architecture. See `config(8)`, for details on configuring a new SunOS kernel.

The manual page for a device driver may also include a DIAGNOSTICS section, listing error messages that the driver might produce. Normally, these messages are logged to the appropriate system log using the kernel's standard message-buffering mechanism (see `syslogd(8)`); they may also appear on the system console.

**Ioctls**

Various special functions, such as querying or altering the operating characteristics of a device, are performed by supplying appropriate parameters to the `ioctl(2)` system call. These parameters are often referred to as "ioctls." Ioctls for a specific device are presented in the manual page for that device. Ioctls that pertain to a class of devices are listed in a manual page with a name that suggests the class of device, and ending in 'io', such as `mtio(4)` for magnetic tape devices, or `dkio(4S)` for disk controllers. In addition, some ioctls operate directly on higher-level objects such as files, terminals, sockets, and streams:

- Ioctls that operate directly on files, file descriptors, and sockets are described in `filio(4)`. Note: the `fcntl(2)` system call is the primary method for operating on file descriptors as such, rather than on the underlying files. Also note that the `setsockopt` system call (see `getsockopt(2)`) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. Ioctls for a specific network interface are documented in the manual page for that interface.
- Ioctls for terminals, including pseudo-terminals, are described in `termio(4)`. This manual page includes information about both the BSD `termios` structure, as well as the System V `termio` structure.
- Ioctls for STREAMS are described in `streamio(4)`.

**Devices Always Present**

Device drivers present in every kernel include:

- The paging device; see `drum(4)`.
- Drivers for accessing physical, virtual, and I/O space in memory; see `mem(4S)`.
- The data sink; see `null(4)`.



**Terminals and Serial Communications Devices**

Serial communication lines are normally supported by the terminal driver; see `tty(4)`. This driver manages serial lines provided by communications drivers, such as those described in `mti(4S)` and `zs(4S)`. The terminal driver also handles serial lines provided by virtual terminals, such as the Sun console monitor described in `console(4S)`, and true pseudo-terminals, described in `pty(4)`.

**Disk Devices**

Drivers for the following disk controllers provide standard block and raw interfaces under SunOS;

- SCSI controllers, in `sd(4S)`,
- Xylogics 450 and 451 SMD controllers, in `xy(4S)`,
- Xylogics 7053 SMD controllers, in `xd(4S)`.

Ioctls to query or set a disk's geometry and partitioning are described in `dkio(4S)`.

**Magnetic Tape Devices**

Magnetic tape devices supported by SunOS include those described in `ar(4S)`, `tm(4S)`, `st(4S)`, and `xt(4S)`. Ioctls for all tape-device drivers are described in `mtio(4S)`.

**Frame Buffers**

Frame buffer devices include color frame buffers described in the `cg*(4S)` manual pages, monochrome frame buffers described in the `bw*(4S)` manual pages, graphics processor interfaces described in the `gp*(4S)` manual pages, and an indirect device for the console frame buffer described in `fb(4S)`. Ioctls for all frame-buffer devices are described in `fbio(4S)`.

**Miscellaneous Devices**

Miscellaneous devices include the console keyboard described in `kbd(4S)`, the console mouse described in `mouse(4S)`, window devices described in `win(4S)`, and the DES encryption-chip interface described in `des(4S)`.

**Network-Interface Devices**

SunOS supports the 10-Megabit Ethernet as its primary network interface; see `ec(4S)`, `ie(4S)`, and `le(4S)` for details. However, a software loopback interface, `lo(4)` is also supported. General properties of these network interfaces are described in `if(4N)`, along with the ioctls that operate on them.

Support for network routing is described in `routing(4N)`.

**Protocols and Protocol Families**

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in `inet(4F)`, is the primary protocol family primary supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the `AF_INET` address family when binding a socket; see `socket(2)` for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in `ip(4P)`. This is the default protocol for `SOCK_RAW` type sockets within the `AF_INET` domain.
- The Transmission Control Protocol (TCP); see `tcp(4P)`. This is the default protocol for `SOCK_STREAM` type sockets.
- The User Datagram Protocol (UDP); see `udp(4P)`. This is the default protocol for `SOCK_DGRAM` type sockets.
- The Address Resolution Protocol (ARP); see `arp(4P)`.
- The Internet Control Message Protocol (ICMP); see `icmp(4P)`.

The Network Interface Tap (NIT) protocol, described in `nit(4P)`, is a STREAMS-based facility for accessing the network at the link level.

## SEE ALSO

`fcntl(2)`, `getsockopt(2)`, `ioctl(2)`, `socket(2)`, `ar(4S)`, `arp(4P)`, `dkio(4S)`, `drum(4)`, `ec(4S)`, `fb(4S)`, `fbio(4S)`, `filio(4)`, `icmp(4P)`, `if(4N)`, `inet(4F)`, `ip(4P)`, `kbd(4S)`, `le(4)`, `lo(4)`, `mbio(4S)`, `mem(4S)`, `mti(4)`, `mtio(4)`, `nit(4P)`, `null(4)`, `pty(4)`, `routing(4N)`, `sd(4S)`, `st(4S)`, `streamio(4)`, `tcp(4P)`, `termio(4)`, `tm(4S)`, `tty(4)`, `udp(4P)`, `win(4S)`, `xd(4S)`, `xy(4S)`, `zs(4S)`

## LIST OF DEVICES, INTERFACES AND PROTOCOLS

| Name                 | Appears on Page          | Description                                                          |
|----------------------|--------------------------|----------------------------------------------------------------------|
| <code>alm</code>     | <code>mcp(4S)</code>     | Asynchronous Line Multiplexer                                        |
| <code>ar</code>      | <code>ar(4S)</code>      | Archive 1/4 inch Streaming Tape Drive                                |
| <code>arp</code>     | <code>arp(4P)</code>     | Address Resolution Protocol                                          |
| <code>bk</code>      | <code>bk(4)</code>       | line discipline for machine-machine communication                    |
| <code>bwone</code>   | <code>bwone(4S)</code>   | Sun-1 black and white frame buffer                                   |
| <code>bwtwo</code>   | <code>bwtwo(4S)</code>   | Sun-3/Sun-2 black and white frame buffer                             |
| <code>cgeight</code> | <code>cgeight(4S)</code> | 24-bitcolor memory frame buffer                                      |
| <code>cgfour</code>  | <code>cgfour(4S)</code>  | Sun-3 color memory frame buffer                                      |
| <code>cgone</code>   | <code>cgone(4S)</code>   | Sun-1 color graphics interface                                       |
| <code>cgsix</code>   | <code>cgsix(4S)</code>   | Sun-3, Sun-4, and Sun-3x low-end graphics accelerator                |
| <code>cgthree</code> | <code>cgthree(4S)</code> | Sun386i color memory frame buffer                                    |
| <code>cgtwo</code>   | <code>cgtwo(4S)</code>   | Sun-3/Sun-2 color graphics interface                                 |
| <code>clone</code>   | <code>clone(4)</code>    | open any minor device on a STREAMS driver                            |
| <code>console</code> | <code>console(4S)</code> | console driver and terminal emulator for the Sun workstation         |
| <code>des</code>     | <code>des(4S)</code>     | DES encryption chip interface                                        |
| <code>dkio</code>    | <code>dkio(4S)</code>    | generic disk control operations                                      |
| <code>drum</code>    | <code>drum(4)</code>     | paging device                                                        |
| <code>ec</code>      | <code>ec(4S)</code>      | 3Com 10 Mb/s Ethernet interface                                      |
| <code>fb</code>      | <code>fb(4S)</code>      | driver for Sun console frame buffer                                  |
| <code>fbio</code>    | <code>fbio(4S)</code>    | general properties of frame buffers                                  |
| <code>fd</code>      | <code>fd(4S)</code>      | Disk driver for Floppy Disk Controllers                              |
| <code>filio</code>   | <code>filio(4)</code>    | ioctls that operate directly on files, file descriptors, and sockets |
| <code>fpa</code>     | <code>fpa(4S)</code>     | Sun-3 floating point accelerator                                     |
| <code>gpone</code>   | <code>gpone(4S)</code>   | Sun-3/Sun-2 graphics processor                                       |
| <code>icmp</code>    | <code>icmp(4P)</code>    | Internet Control Message Protocol                                    |
| <code>ie</code>      | <code>ie(4S)</code>      | Intel 10 Mb/s Ethernet interface                                     |
| <code>if</code>      | <code>if(4N)</code>      | general properties of network interfaces                             |
| <code>inet</code>    | <code>inet(4F)</code>    | Internet protocol family                                             |
| <code>ip</code>      | <code>ip(4P)</code>      | Internet Protocol                                                    |
| <code>kb</code>      | <code>kb(4M)</code>      | Sun keyboard STREAMS module                                          |
| <code>kbd</code>     | <code>kbd(4S)</code>     | Sun keyboard                                                         |
| <code>kmem</code>    | <code>mem(4S)</code>     | main memory and bus I/O space                                        |
| <code>ldterm</code>  | <code>ldterm(4M)</code>  | standard terminal STREAMS module                                     |
| <code>le</code>      | <code>le(4S)</code>      | Sun-3/50, Sun-3/60 10MB Ethernet interface                           |
| <code>lo</code>      | <code>lo(4)</code>       | software loopback network interface                                  |
| <code>lofs</code>    | <code>lofs(4S)</code>    | loopback virtual file system                                         |
| <code>mbio</code>    | <code>mem(4S)</code>     | main memory and bus I/O space                                        |
| <code>mbmem</code>   | <code>mem(4S)</code>     | main memory and bus I/O space                                        |
| <code>mcp</code>     | <code>mcp(4S)</code>     | MCP Multiprotocol Communications Processor                           |
| <code>mem</code>     | <code>mem(4S)</code>     | main memory and bus I/O space                                        |
| <code>mouse</code>   | <code>mouse(4S)</code>   | Sun mouse                                                            |
| <code>ms3</code>     | <code>mouse(4S)</code>   | Sun mouse                                                            |
| <code>ms</code>      | <code>ms(4M)</code>      | Sun mouse STREAMS module                                             |



|          |              |                                                          |
|----------|--------------|----------------------------------------------------------|
| mti      | mti(4S)      | Systech MTI-800/1600 multi-terminal interface            |
| mtio     | mtio(4)      | UNIX system magnetic tape interface                      |
| NFS      | nfs(4P)      | network file system                                      |
| nif_pf   | nit_pf(4M)   | streams NIT packet filtering module                      |
| nit      | nit(4P)      | Network Interface Tap facility                           |
| nit_buf  | nit_buf(4M)  | streams NIT buffering module                             |
| nit_if   | nit_if(4M)   | streams NIT device interface module                      |
| null     | null(4)      | data sink                                                |
| pp       | pp(4)        | Centronics-compatible parallel printer port              |
| pty      | pty(4)       | pseudo terminal driver                                   |
| root     | root(4S)     | pseudo-driver for Sun root disk                          |
| routing  | routing(4N)  | system supporting for local network packet routing       |
| sd       | sd(4S)       | Disk driver for SCSI Disk Controllers                    |
| sockio   | sockio(4)    | ioctl's that operate directly on sockets                 |
| st       | st(4S)       | Sysgen SC 4000 and Emulex MT-02 Tape Controller          |
| streamio | streamio(4)  | STREAMS ioctl commands                                   |
| tcp      | tcp(4P)      | Transmission Control Protocol                            |
| termio   | termio(4)    | general terminal interface                               |
| tm       | tm(4S)       | tapemaster 1/2 inch tape drive                           |
| ttcompat | ttcompat(4M) | V7/4BSD compatibility STREAMS module                     |
| tty      | tty(4)       | controlling terminal interface                           |
| udp      | udp(4P)      | User Datagram Protocol                                   |
| vme16d16 | mem(4S)      | main memory and bus I/O space                            |
| vme16d32 | mem(4S)      | main memory and bus I/O space                            |
| vme24d16 | mem(4S)      | main memory and bus I/O space                            |
| vme24d32 | mem(4S)      | main memory and bus I/O space                            |
| vme32d16 | mem(4S)      | main memory and bus I/O space                            |
| vme32d32 | mem(4S)      | main memory and bus I/O space                            |
| vp       | vp(4S)       | Ikon 10071-5 Versatec parallel printer interface         |
| vpc      | vpc(4S)      | Systech VPC-2200 Versatec plotter and Centronics printer |
| win      | win(4S)      | Sun window system                                        |
| xd       | xd(4S)       | Disk driver for Xylogics 7053 SMD Disk Controller        |
| xt       | xt(4S)       | Xylogics 472 1/2 inch tape controller                    |
| xy       | xy(4S)       | Disk driver for Xylogics SMD Disk Controllers            |
| zero     | zero(4S)     | source of zeroes                                         |
| zs       | zs(4S)       | Zilog 8530 SCC serial communications driver              |

**NAME**

**bwone** – Sun-1 black and white frame buffer

**CONFIG — SUN-2 SYSTEM**

**device bwone0 at mbmem ? csr 0xc0000 priority 3**

**DESCRIPTION**

The **bwone** interface provides access to Sun-1 system black and white graphics controller boards. It supports the ioctls described in **fbio(4S)**.

**FILES**

**/dev/bwone[0-9]**      device files

**SEE ALSO**

**mmap(2)**, **fb(4S)**, **fbio(4S)**

**BUGS**

Use of vertical-retrace interrupts is not supported.

The video state returned by the **FBIIOGVVIDEO** ioctl may be incorrect. It is not possible for the driver to determine the state of the hardware video enable bit, so it reports the last state stored by the **FBIOSVIDEO** ioctl. User processes which map the frame buffer can directly enable or disable the video, unknown to the driver.



**NAME**

bwtwo – Sun-3/Sun-2 black and white frame buffer

**CONFIG — SUN-3/SUN-3x SYSTEMS**

```
device bwtwo0 at obmem 1 csr 0xff000000 priority 4
device bwtwo0 at obmem 2 csr 0x100000 priority 4
device bwtwo0 at obmem 3 csr 0xff000000 priority 4
device bwtwo0 at obmem 4 csr 0xff000000
device bwtwo0 at obmem 7 csr 0xff000000 priority 4
device bwtwo0 at obmem ? csr 0x50300000 priority 4
```

The first synopsis line given above is used to generate a kernel for Sun-3/75, Sun-3/140 or Sun-3/160 systems; the second, for a Sun-3/50 system; the third, for a Sun-3/260 system; the fourth, for a Sun-3/110 system; the fifth, for a Sun-3/60 system; and the sixth for Sun-3/80 and Sun-3/470 systems.

**CONFIG — SUN-2 SYSTEM**

```
device bwtwo0 at obmem 1 csr 0x700000 priority 4
device bwtwo0 at obio 2 csr 0x0 priority 4
```

The first synopsis line given above is used to generate a kernel for a Sun-2/120 or Sun-2/170 system; the second, for a Sun-2/50 or Sun-2/160 system.

**CONFIG — Sun386i SYSTEM**

```
device bwtwo0 at obmem ? csr 0xA0200000
```

**DESCRIPTION**

The **bwtwo** interface provides access to Sun monochrome memory frame buffers. It supports the ioctl's described in **fbio(4S)**.

If **flags 0x1** is specified, frame buffer write operations are buffered through regular high-speed RAM. This "copy memory" mode of operation speeds frame buffer accesses, but consumes an extra 128K bytes of memory. Only Sun-2, Sun-3/75, and Sun-3/160 systems support copy memory; on other systems a warning message is printed and the flag is ignored.

Reading or writing to the frame buffer is not allowed — you must use the **mmap(2)** system call to map the board into your address space.

**FILES**

**/dev/bwtwo[0-9]**          device files

**SEE ALSO**

**mmap(2)**, **cgfour(4S)**, **fb(4S)**, **fbio(4S)**

**BUGS**

Use of vertical-retrace interrupts is not supported.

**NAME**

**cgeight** – 24-bit color memory frame buffer

**CONFIG — SUN-3 AND SUN-4 SYSTEMS**

**device cgeight0 at obmem 7 csr 0xff300000 priority 4**

**device cgeight0 at obio 4 csr 0xfb300000 priority 4**

The first synopsis line should be used to generate a kernel for the Sun-3/60; the second synopsis for a Sun-4/110 or Sun-4/150 system.

**CONFIG — SUN-3x SYSTEM**

**device cgeight0 at obio ? csr 0x50300000 priority 4**

**DESCRIPTION**

The **cgeight** is a 24-bit color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented optionally on the Sun-4/110, Sun-4/150, Sun-3/60, Sun-3/470 and Sun-3/80 system models. It provides the standard frame buffer interface as defined in **fbio(4S)**.

In addition to the **ioctl**s described under **fbio(4S)**, the **cgeight** interface responds to two **cgeight**-specific colormap **ioctl**s, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the **ioctl** return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the **red**, **green**, and **blue** members of its **fbcmmap** structure argument; **fbcmmap** is defined in **/usr/include/sun/fbio.h** as:

```
struct fbcmmap {
    int      index;      /* first element (0 origin) */
    int      count;      /* number of elements */
    unsigned char *red;   /* red color map elements */
    unsigned char *green; /* green color map elements */
    unsigned char *blue;  /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The systems have an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX\_GROUP** macro (see **/usr/include/pixrect/pr\_planegroups.h**).

When using the **mmap** system call to map in the **cgeight** frame buffer. The device looks like:

|                          |                                    |                    |
|--------------------------|------------------------------------|--------------------|
| <b>DACBASE: 0x200000</b> | <b>-&gt; Brooktree Ramdac</b>      | <b>16 bytes</b>    |
| <b>0x202000</b>          | <b>-&gt; P4 Register</b>           | <b>4 bytes</b>     |
| <b>OVLBASE: 0x210000</b> | <b>-&gt; Overlay Plane</b>         | <b>1152x900x1</b>  |
| <b>0x230000</b>          | <b>-&gt; Overlay Enable Planea</b> | <b>1152x900x1</b>  |
| <b>0x250000</b>          | <b>-&gt; 24-bit Frame Buffera</b>  | <b>1152x900x32</b> |

**FILES**

**/dev/cgeight0**

**/usr/include/sun/fbio.h**

**/usr/include/pixrect/pr\_planegroups.h**

**SEE ALSO**

**mmap(2)**, **fbio(4S)**





**NAME**

**cgfour** – Sun-3 color memory frame buffer

**CONFIG — SUN-3 SYSTEM**

**device cgfour0 at obmem 4 csr 0xff000000 priority 4**

**device cgfour0 at obmem 7 csr 0xff000000 priority 4**

The first synopsis line given should be used to generate a kernel for the Sun-3/110 system; and the second, for a Sun-3/60 system.

**CONFIG — SUN-3x SYSTEM**

**device cgfour0 at obmem ? csr 0x50300000 priority 4**

**DESCRIPTION**

The **cgfour** is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented on the Sun-3/110 system and some Sun-3/60 system models. It provides the standard frame buffer interface as defined in **fbio(4S)**.

In addition to the **ioctl**s described under **fbio(4S)**, the **cgfour** interface responds to two **cgfour**-specific colormap **ioctl**s, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the **ioctl** return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the **red**, **green**, and **blue** members of its **fbcmmap** structure argument; **fbcmmap** is defined in **<sun/fbio.h>** as:

```
struct fbcmmap {
    int      index;      /* first element (0 origin) */
    int      count;      /* number of elements */
    unsigned char *red;   /* red color map elements */
    unsigned char *green; /* green color map elements */
    unsigned char *blue;  /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The Sun-3/60 system has an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX\_GROUP** macro (see **<pixrect/pr\_planegroups.h>**).

**FILES**

**/dev/cgfour0**

**SEE ALSO**

**mmap(2)**, **fbio(4S)**



**NAME**

cgone – Sun-1 color graphics interface

**CONFIG — SUN-2 SYSTEM**

device cgone0 at mbmem ? csr 0xec000 priority 3

**DESCRIPTION**

The **cgone** interface provides access to the Sun-1 system color graphics controller board, which is normally supplied with a 13" or 19" RS170 color monitor. It provides the standard frame buffer interface as defined in **fbio(4S)**.

It supports the **FBIOPIXRECT** ioctl which allows SunView to be run on it; see **fbio(4S)**

The hardware consumes 16 kilobytes of Multibus memory space. The board starts at standard addresses 0xE8000 or 0xEC000. The board must be configured for interrupt level 3.

**FILES**

/dev/cgone[0-9]

**SEE ALSO**

**mmap(2)**, **fbio(4S)**

**BUGS**

Use of color board vertical-retrace interrupts is not supported.

**NAME**

**cgsix** — low-end graphics accelerator with 8-bit color frame buffer

**CONFIGURATION — SUN-3, SUN-4, and SUN-3x SYSTEMS**

**device cgsix0 at obmem ? csr 0xff000000 priority 4**

**device cgsix0 at obmem ? csr 0xfb000000 priority 4**

**device cgsix0 at obmem ? csr 0x50000000 priority 4**

The first synopsis line is used for Sun-3 systems, the second for Sun-4 systems, and the third for Sun-3x systems.

**DESCRIPTION**

**cgsix(4S)** is a low-end, P4 graphics accelerator that increases vector and polygon drawing performance. It uses an 8-bit color frame buffer and provides the standard frame buffer interface, as defined in **fbio(4S)**.

In addition to the **ioctl**s described under **fbio**, the **cgsix** interface responds to two **cgsix**-specific colormap **ioctl**s, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the **ioctl** return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the **red**, **green**, and **blue** members of its **fbcmmap** structure argument; **fbcmmap** is defined in **/usr/include/sun/fbio.h** as:

```
struct fbcmmap {
    int      index; /* first element (0 origin) */
    int      count; /* number of elements */
    unsigned char *red; /* red color map elements */
    unsigned char *green; /* green color map elements */
    unsigned char *blue; /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

**cgsix** contains memory that may be mapped in through calls to the **cgsixmmap()** function. Each portion of this memory is specified by an offset from the base address that is configured into the kernel. The portions that may be mapped are the colormap, the **FBC/TEC**, the **FHC/THC**, and the framebuffer. The exact offsets are defined in **/usr/includesundev/cg6reg.h**.

**FILES**

**/dev/cgsix0**

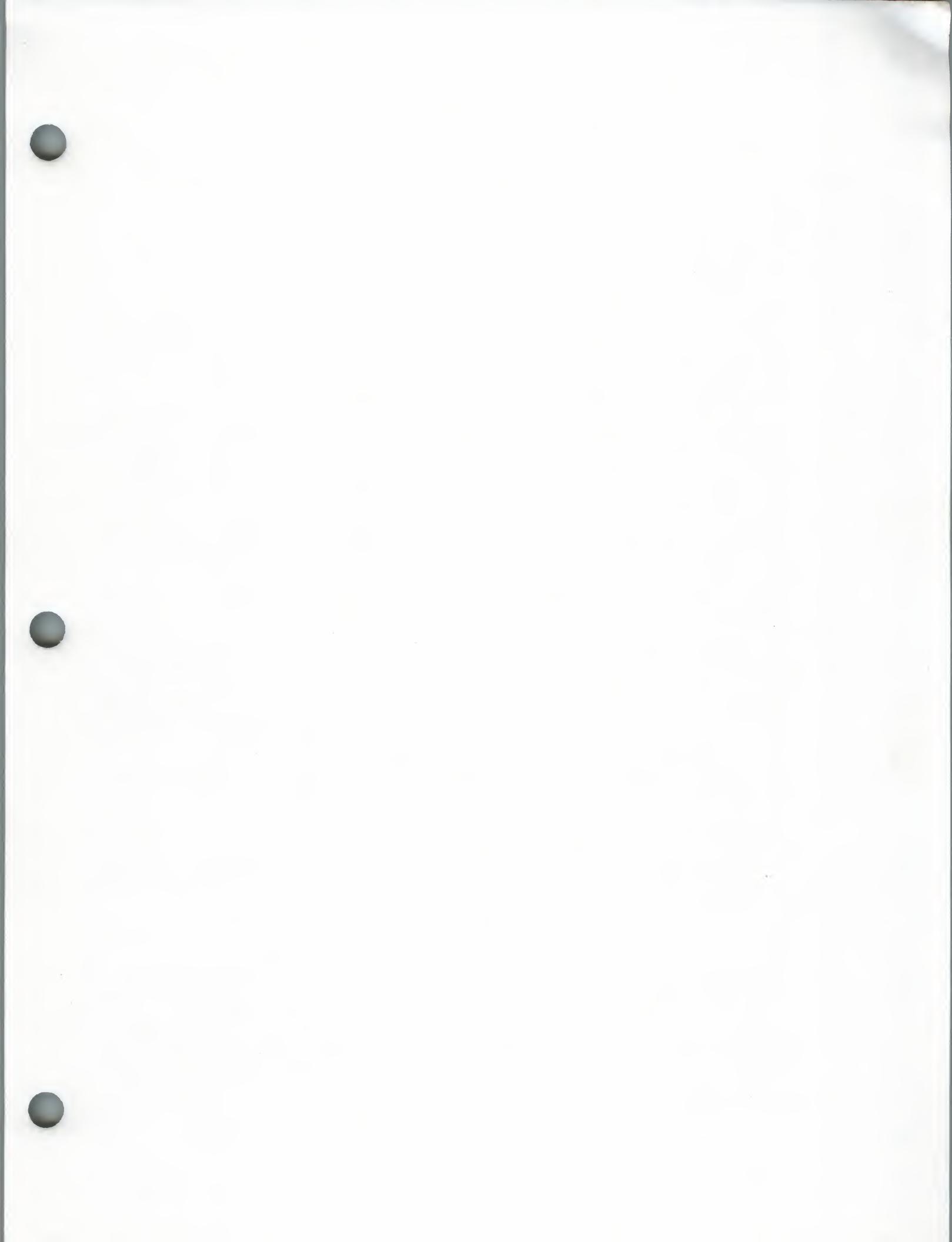
**/usr/include/sundev/cg6reg.h**

**SEE ALSO**

**mmap(2)**, **fbio(4S)**









**NAME**

console – console driver and terminal emulator for the Sun workstation

**CONFIG**

None; included in standard system.

**SYNOPSIS**

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/console", mode);
```

**DESCRIPTION**

console is an indirect driver for the Sun console terminal. On a Sun workstation, this driver refers to the workstation console driver, which implements a standard UNIX system terminal. On a Sun server without a keyboard or a frame buffer, this driver refers to the CPU serial port driver (zs(4S)); a terminal is normally connected to this port.

The workstation console does not support any of the **termio(4)** device control functions specified by flags in the **c\_cflag** word of the **termios** structure or by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c\_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio(4)** functions must be performed by **STREAMS** modules pushed atop the driver; when a slave device is opened, the **ldterm(4M)** and **ttcompat(4M)** **STREAMS** modules are automatically pushed on top of the stream, providing the standard **termio(4)** interface.

The workstation console driver calls the PROM resident monitor to output data to the console frame buffer. Keystrokes from the CPU serial port to which the keyboard is connected are routed through the keyboard **STREAMS** module (**kb(4M)**) and treated as input.

When the Sun window system **win(4S)** is active, console input is directed through the window system rather than being treated as input by the workstation console driver.

**IOCTLS**

An **ioctl** **TIOCCONS** can be applied to pseudo-terminals (**pty(4)**) to route output that would normally appear on the console to the pseudo-terminal instead. Thus, the window system does a **TIOCCONS** on a pseudo-terminal so that the system will route console output to the window to which that pseudo-terminal is connected, rather than routing output through the PROM monitor to the screen, since routing output through the PROM monitor destroys the integrity of the screen. Note: when you use **TIOCCONS** in this way, the console *input* is routed from the pseudo-terminal as well.

If a **TIOCCONS** is performed on **/dev/console**, or the pseudo-terminal to which console output is being routed is closed, output to the console will again be routed to the workstation console driver.

**ANSI STANDARD TERMINAL EMULATION**

The Sun Workstation's PROM monitor provides routines that emulates a standard ANSI X3.64 terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun terminal emulator and the VT100 are *not* compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with scrolling, (x, y) cursor addressability, and a number of other control functions.

The Sun console displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see **CTRL-J** below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.



### Control Sequence Syntax

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation

CTRL-*X*

for some character *X*, represents a control character.

Other ANSI control sequences are of the form

ESC [*params**char*

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC represents the ASCII escape character (ESC, CTRL-[, 0x1B).

[ The next character is a left square bracket '[' (0x5B).

*params* are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

*char* represents a function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character 'Escape'):

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| ESC[m              | <i>select graphic rendition with default parameter</i> |
| ESC[7m             | <i>select graphic rendition with reverse image</i>     |
| ESC[33;54H         | <i>set cursor position</i>                             |
| ESC[123;456;0;;3;B | <i>move cursor down</i>                                |

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last *n* are used, where *n* is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as 'ESC[5M') is *not* equivalent to ESC[5;M (interpreted as 'ESC[5;1M') which is ultimately interpreted as 'ESC[1M').

In the syntax descriptions below, parameters are represented as '#' or '#1;#2'.

### ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the SUNRESET escape sequence.



**Control Character Functions****CTRL-G (0x7) Bell (BEL)**

The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It 'rings the bell' by flashing the entire screen. The Sun-2 models have an audible bell which beeps. The window system flashes the window.

**CTRL-H (0x8) Backspace (BS)**

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

**CTRL-I (0x9) Tab (TAB)**

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

**CTRL-J (0xA) Line-feed (LF)**

The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or "wraps around" depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, 'wrap-around' mode is entered. 'ESC [ 1 r' exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. When any line-feed occurs, the line that the cursor moves to is cleared. This means that no scrolling occurs. Wrap-around mode is not implemented in the window system.

The screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place and the console is in normal scroll mode ('ESC [ 1 r'), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when any control character from the set {VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US} is found. At that point, the screen is scrolled by *N* lines ( $N \geq 1$ ) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the 'Set scrolling' (ESC[r) control function below.

**CTRL-K (0xB) Reverse Line-feed**

The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.

**CTRL-L (0xC) Form-feed (FF)**

The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.

**CTRL-M (0xD) Return (CR)**

The cursor moves to the leftmost character position on the current line.

**Escape Sequence Functions****CTRL-[ (0x1B) Escape (ESC)**

This is the escape character. Escape initiates a multi-character control sequence.

**ESC[#@ Insert Character (ICH)**

Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.



- ESC[#A           Cursor Up (CUU)  
Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.
- ESC[#B           Cursor Down (CUD)  
Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.
- ESC[#C           Cursor Forward (CUF)  
Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.
- ESC[#D           Cursor Backward (CUB)  
Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.
- ESC[#E           Cursor Next Line (CNL)  
Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.
- ESC[#1;#2f       Horizontal And Vertical Position (HVP)  
or  
ESC[#1;#2H       Cursor Position (CUP)  
Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.
- ESC[J            Erase in Display (ED)  
Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.
- ESC[K            Erase in Line (EL)  
Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.
- ESC[#L           Insert Line (IL)  
Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with spaces; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.
- ESC[#M           Delete Line (DL)  
Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with spaces; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.
- ESC[#P           Delete Character (DCH)  
Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.



**ESC[#m                    Select Graphic Rendition (SGR)**

Takes one parameter, # (default 0). Note: unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0 Normal rendition.

7 Negative (reverse) image.

Negative image displays characters as white-on-black if the screen mode is currently black-on-white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and selects the negative image rendition.

**ESC[p                    Black On White (SUNBOW)**

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

**ESC[q                    White On Black (SUNWOB)**

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

**ESC[#r                    Set scrolling (SUNSCRL)**

Takes one parameter, # (default 0). Sets to # an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of "jump" when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates "wrap mode" instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever occurs. 'ESC [ 1 r' exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

**ESC[s                    Reset terminal emulator (SUNRESET)**

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are

**4014 TERMINAL EMULATION**

The PROM monitor for Sun models 100U and 150U provides the Sun Workstation with the capability to emulate a subset of the Tektronix 4014 terminal. This feature does not exist in other Sun PROMs and will be removed from models 100U and 150U in future Sun releases. `tektool(1)` provides Tektronix 4014 terminal emulation and should be used instead of relying on the capabilities of the PROM monitor.

**FILES**

`/dev/console`

**SEE ALSO**

`tektool(1)` `kb(4M)`, `pty(4)`, `termio(4)`, `ttcompat(4M)`, `ldterm(4M)`, `win(4S)`, `zs(4S)`

ANSI Standard X3.64, "Additional Controls for Use with ASCII", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

**BUGS**

TIOCCONS should be restricted to the owner of `/dev/console`.





**NAME**

db – SunDials STREAMS module

**CONFIG**

pseudo-device**dbn**

**SYNOPSIS**

```
#include <sys/stream.h>
#include <sundev/vuid_event.h>
#include <sundev/dbio.h>
#include <sys/time.h>
#include <sys/ioctl.h>
open("/dev/dialbox", O_RDWR);
ioctl(fd, I_PUSH, "db");
```

**DESCRIPTION**

The db STREAMS module processes the byte streams generated by the SunDials dial box. This dial box generates a stream of bytes, which encode the identity of the dials and the amount by which they are turned.

Each dial sample in the byte stream consists of three bytes. The first byte identifies which dial was turned and the next two bytes return the delta in signed binary format. An event from a dial is constrained to lie between 0x80 and 0x87. When bound to an application using the window system, *Virtual User Input Device* events are generated.

A stream with db pushed into it can emit *firm\_events* as specified by the protocol of a VUID. db understands the VUIDSFORMAT and VUIDGFORMAT ioctls (see reference below), as defined in /usr/include/sundev/dbio.h and /usr/include/sundev/vuid\_event.h. All other ioctls are passed downstream. db sets the parameters of a serial port when it is opened. No *termios(4)* ioctls should be performed on a db stream, as db expects the device parameters to remain as it set them.

**IOCTLS**

VUIDSFORMAT

VUIDGFORMAT

These are standard *Virtual User Input Device* ioctls. See *SunView 1 System Programmer's Guide* for a description of their operation.

**FILES**

```
/usr/include/sundev/dbio.h
/usr/include/sundev/vuid_event.h
/usr/include/sys/ioctl.h
/usr/include/sys/stream.h
/usr/include/sys/time.h
```

**SEE ALSO**

*termios(4)*, *dbconfig(6)*, *dialtest(6)*

*SunView 1 System Programmer's Guide*, *SunDials Programmers Guide*

**BUGS**

VUIDSADDR and VUIDGADDR are not supported.

**WARNING**

The SunDials dial box must be used with a serial port.



**NAME**

des – DES encryption chip interface

**CONFIG — SUN-3 SYSTEM**

device des0 at obio ? csr 0x1c0000

**CONFIG — SUN-3x SYSTEM**

device des0 at obio ? csr 0x66002000

**CONFIG — SUN-2 SYSTEM**

des0 at virtual ? csr 0xee1800

**SYNOPSIS**

#include <sys/des.h>

**DESCRIPTION**

The **des** driver provides a high level interface to the AmZ8068 Data Ciphering Processor, a hardware implementation of the NBS Data Encryption Standard.

The high level interface provided by this driver is hardware independent and could be shared by future drivers in other systems.

The interface allows access to two modes of the DES algorithm: Electronic Code Book (ECB) and Cipher Block Chaining (CBC). All access to the DES driver is through **ioctl(2)** calls rather than through reads and writes; all encryption is done in-place in the user's buffers.

**IOCTLS**

The **ioctls** provided are:

**DESIOCBLOCK**

This call encrypts/decrypts an entire buffer of data, whose address and length are passed in the '**struct desparams**' addressed by the argument. The length must be a multiple of 8 bytes.

**DESIOCQUICK**

This call encrypts/decrypts a small amount of data quickly. The data is limited to **DES\_QUICKLEN** bytes, and must be a multiple of 8 bytes. Rather than being addresses, the data is passed directly in the '**struct desparams**' argument.

**FILES**

/dev/des

**SEE ALSO**

**des(1)**, **des\_crypt(3)**

*Federal Information Processing Standards Publication 46*

*AmZ8068 DCP Product Description, Advanced Micro Devices*





## NAME

fbio – general properties of frame buffers

## DESCRIPTION

All of the Sun frame buffers support the same general interface. Each responds to a **FBIOGTYPE** ioctl which returns information in a structure defined in `<sun/fbio.h>`:

```

struct fbtype {
    int    fb_type;        /* as defined below */
    int    fb_height;      /* in pixels */
    int    fb_width;       /* in pixels */
    int    fb_depth;       /* bits per pixel */
    int    fb_cmsize;      /* size of color map (entries) */
    int    fb_size;        /* total size in bytes */
};

#define FBTYPE_SUN1BW      0
#define FBTYPE_SUN1COLOR  1
#define FBTYPE_SUN2BW      2
#define FBTYPE_SUN2COLOR  3
#define FBTYPE_SUN2GP      4
#define FBTYPE_SUN5COLOR  5
#define FBTYPE_SUN3COLOR  6
#define FBTYPE_MEMCOLOR    7
#define FBTYPE_SUN4COLOR  8

```

Each device has an **FBTYPE** which is used by higher-level software to determine how to perform raster-op and other functions. Each device is used by opening it, doing an **FBIOGTYPE** ioctl to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

Full-fledged frame buffers (that is, those that run SunView) implement an **FBIOGPIXRECT** ioctl, which returns a **pixrect**. This call is made only from inside the kernel. The returned **pixrect** is used by **win(4S)** for cursor tracking and colormap loading.

**FBIOVIDEO** and **FBIOGVIDEO** are general-purpose ioctls for controlling possible video features of frame buffers. They are defined in `<sun/fbio.h>`. These ioctls either set or return the value of a flags integer. At this point, only the **FBVIDEO\_ON** option is available, controlled by **FBIOVIDEO**. **FBIOGVIDEO** returns the current video state.

The **FBIOATTR** and **FBIOGATTR** ioctls allow access to special features of newer frame buffers. They use the following structures as defined in `<sun/fbio.h>`:

```

#define FB_ATTR_NDEVSPECIFIC 8      /* no. of device specific values */
#define FB_ATTR_NEMUTYPES 4      /* no. of emulation types */
struct fbsattr {
    int    flags;           /* misc flags */
#define FB_ATTR_AUTOINIT 1      /* emulation auto init flag */
#define FB_ATTR_DEVSPECIFIC 2  /* dev. specific stuff valid flag */
    int    emu_type;        /* emulation type (-1 if unused) */
    int    dev_specific[FB_ATTR_NDEVSPECIFIC]; /* catchall */
};
struct fbgattr {
    int    real_type;       /* real device type */
    int    owner;           /* PID of owner, 0 if myself */
    struct fbtype fbtype;    /* fbtype info for real device */
    struct fbsattr sattr;    /* see above */
    int    emu_types[FB_ATTR_NEMUTYPES]; /* possible emulations */
                                           /* (-1 if unused) */
};

```

**SEE ALSO**

**mmap(2), bwone(4S), bwtwo(4S), cgeight(4S), cgfour(4S), cgone(4S), cgtwo(4S), fb(4S), gpone(4S), win(4S)**

**BUGS**

**FBIOSATTR** and **FBIOGATTR** are only supported by the **cgfour(4S)** and **cgeight(4S)** frame buffers.

The **FBVIDEO\_ON** flag may be incorrect for Sun-1 system black and white frame buffers; see **bwone(4S)**.



**NAME**

fd – Disk driver for Floppy Disk Controllers

**CONFIG — Sun386i SYSTEMS**

controller fdc0 at atmem ? csr 0x1000 dmachan 2 irq 6 priority 2

disk fd0 at fdc0 drive 0 flags 0

**CONFIG — SUN-3x SYSTEMS**

controller fdc0 at obio ? csr 0x6e000000 priority 6 vector fdintr 0x5c

This synopsis should be used to generate a kernel for a Sun-3/80 system only.

**AVAILABILITY**

Sun386i and Sun-3/80 systems only.

**DESCRIPTION**

The block-files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

**Disk Support**

The **fd0** partition on a floppy disk is normally used for the file system.

**FILES**

1.44 MB Floppy Disk Drives:

/dev/fd0a                      block file

/dev/fd0c                      block file /dev/rfd0a raw file /dev/rfd0c raw file

720 K Floppy Disk Drives:

/dev/fdl0a                      block file

/dev/fdl0c                      block file

/dev/rfdl0a                      raw file

/dev/rfdl0c                      raw file

**SEE ALSO**

dkio(4S)

**DIAGNOSTICS**

**fd drv %d, trk %d: %s**

A command such as read or write encountered a format-related error condition. The value of %s is derived from the error number given by the controller, indicating the nature of the error. The track number is relative to the beginning of the partition involved.

**fd drv %d, blk %d: %s**

A command such as read or write encountered an error condition related to I/O. The value of %s is derived from the error number returned by the controller and indicates the nature of the error. The block number is relative to the start of the partition involved.

**fd controller: %s**

An error occurred in the controller. The value of %s is derived from the status returned by the controller and specifies the error encountered.

**fd(%d):%s please insert**

I/O was attempted while the floppy drive door was not latched. The value of %s indicates which disk was expected to be in the drive.

**NAME**

filio – ioctls that operate directly on files, file descriptors, and sockets

**SYNOPSIS**

```
#include <sys/filio.h>
```

**DESCRIPTION**

The IOCTL's listed in this manual page apply directly to files, file descriptors, and sockets, independent of any underlying device or protocol.

Note: the `fcntl(2V)` system call is the primary method for operating on file descriptors as such, rather than on the underlying files.

**IOCTLS for File Descriptors**

|                 |                                                                                                                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FIOCLEX</b>  | The argument is ignored. Set the close-on-exec flag for the file descriptor passed to <code>ioctl</code> . This flag is also manipulated by the <code>F_SETFD</code> command of <code>fcntl(2V)</code> . |
| <b>FIONCLEX</b> | The argument is ignored. Clear the close-on-exec flag for the file descriptor passed to <code>ioctl</code> .                                                                                             |

**IOCTLS for Files**

|                  |                                                                                                                                                                                                                                                                                      |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FIONREAD</b>  | The argument is a pointer to a <code>long</code> . Set the value of that <code>long</code> to the number of immediately readable characters from whatever the descriptor passed to <code>ioctl</code> refers to. This works for files, pipes, sockets, and terminals.                |
| <b>FIONBIO</b>   | The argument is a pointer to an <code>int</code> . Set or clear non-blocking I/O. If the value of that <code>int</code> is a 1 (one) the descriptor is set for non-blocking I/O. If the value of that <code>int</code> is a 0 (zero) the descriptor is cleared for non-blocking I/O. |
| <b>FIOASYNC</b>  | The argument is a pointer to an <code>int</code> . Set or clear asynchronous I/O. If the value of that <code>int</code> is a 1 (one) the descriptor is set for asynchronous I/O. If the value of that <code>int</code> is a 0 (zero) the descriptor is cleared for asynchronous I/O. |
| <b>FIOSETOWN</b> | The argument is a pointer to an <code>int</code> . Set the process-group ID that will subsequently receive <code>SIGIO</code> or <code>SIGURG</code> signals for the object referred to by the descriptor passed to <code>ioctl</code> to the value of that <code>int</code> .       |
| <b>FIOGETOWN</b> | The argument is a pointer to an <code>int</code> . Set the value of that <code>int</code> to the process-group ID that is receiving <code>SIGIO</code> or <code>SIGURG</code> signals for the object referred to by the descriptor passed to <code>ioctl</code> .                    |

**SEE ALSO**

`ioctl(2)`, `fcntl(2V)`, `getsockopt(2)`, `sockio(4)`



**NAME**

**fpa** – Sun-3/Sun-3x floating-point accelerator

**CONFIG — SUN-3/SUN-3X SYSTEM**

**fpa0** at virtual ? csr 0xe0000000

**SYNOPSIS**

```
#include <sundev/fpareg.h>
open("/dev/fpa", flags);
```

**DESCRIPTION**

FPA and FPA+ are compatible floating point accelerators available on certain Sun-3 and Sun-3x systems. They provide hardware contexts for simultaneous use by up to 32 processes. The same **fpa** device driver manages either FPA or FPA+ hardware.

Processes access the device using **open(2)** and **close(2)** system calls, and the FPA is automatically mapped into the process' address space by SunOS. This is normally provided transparently at compile time by a compiler option, such as the **-ffpa** option to **cc(1V)**.

The valid **ioctl(2)** system calls are used only by diagnostics and by system administration programs, such as **fpa\_download(8)**.

**IOCTLS**

|                         |                                                                                                                 |
|-------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>FPA_ACCESS_OFF</b>   | Clear <b>FPA_ACCESS_BIT</b> in FPA state register to disable access to constants RAM using FPA load pointer.    |
| <b>FPA_ACCESS_ON</b>    | Set <b>FPA_ACCESS_BIT</b> in FPA state register to enable access to constants RAM using FPA load pointer.       |
| <b>FPA_FAIL</b>         | Disable the FPA.                                                                                                |
| <b>FPA_GET_DATAREGS</b> | Return the contents of 8 FPA registers.                                                                         |
| <b>FPA_INIT_DONE</b>    | Called when downloading is complete. Allows multiple users to access the FPA.                                   |
| <b>FPA_LOAD_OFF</b>     | Set <b>FPA_LOAD_BIT</b> in FPA state register to disable access to microstore or map RAM via FPA load pointer.  |
| <b>FPA_LOAD_ON</b>      | Set <b>FPA_LOAD_BIT</b> in FPA state register to enable access to microstore or map RAM using FPA load pointer. |

The following two **ioctls** are for diagnostic use only. **fpa** must be compiled with **FPA\_DIAGNOSTICS\_ONLY** defined to enable these two calls.

|                        |                                        |
|------------------------|----------------------------------------|
| <b>FPA_WRITE_STATE</b> | Overwrite the FPA state register.      |
| <b>FPA_WRITE_HCP</b>   | Write to the hard clear pipe register. |

**ERRORS**

The following error messages are returned by **open** system calls only.

|                 |                                                                                        |
|-----------------|----------------------------------------------------------------------------------------|
| <b>EBUSY</b>    | All 32 FPA contexts are being used.                                                    |
| <b>EEXIST</b>   | The current process has already opened <b>/dev/fpa</b> .                               |
| <b>EIO</b>      | Downloading has not completed, so only 1 root process can have the FPA open at a time. |
| <b>ENETDOWN</b> | FPA is disabled.                                                                       |
| <b>ENOENT</b>   | 68881 chip does not exist.                                                             |
| <b>ENXIO</b>    | FPA board does not exist.                                                              |

The following error messages are returned by **ioctl** system calls only.

|               |                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EINVAL</b> | Invalid <b>ioctl</b> . This may occur if diagnostic only <b>ioctls</b> , <b>FPA_WRITE_STATE</b> or <b>FPA_WRITE_HCP</b> , are used with a driver which didn't compile in those calls. |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**EPERM** All ioctl calls except for **FPA\_GET\_DATAREGS** require root execution level.

**EPIPE** The FPA pipe is not clear.

**FILES**

**/dev/fpa** device file for both FPA and FPA+.

**SEE ALSO**

**cc(1V)**, **close(2)**, **ioctl(2)**, **open(2V)**, **fpa\_download(8)**, **fparel(8)**, **fpaversion(8)**

**DIAGNOSTICS**

If hardware problems are detected then all processes with **/dev/fpa** open are killed, and future opens of **/dev/fpa** are disabled.





**NAME**

icmp – Internet Control Message Protocol

**SYNOPSIS**

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
#include <netinet/ip_icmp.h>
```

```
s = socket(AF_INET, SOCK_RAW, proto);
```

**DESCRIPTION**

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed through a “raw socket” for network monitoring and diagnostic functions. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from `getprotobyname` (see `getprotoent(3N)`). ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the `connect(2)` call may also be used to fix the destination for future packets (in which case the `read(2V)` or `recv(2)` and `write(2V)` or `send(2)` system calls may be used).

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the holder of a raw socket with the IP header and options intact.

ICMP is an unreliable datagram protocol layered above IP. It is used internally by the protocol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP “redirect” message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided using the ICMP raw socket.

**ERRORS**

A socket operation may fail with one of the following errors returned:

|               |                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EISCONN       | when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected; |
| ENOTCONN      | when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;                                                                         |
| ENOBUFS       | when the system runs out of memory for an internal data structure;                                                                                                                     |
| EADDRNOTAVAIL | when an attempt is made to create a socket with a network address for which no network interface exists.                                                                               |

**SEE ALSO**

`connect(2)`, `read(2V)`, `recv(2)`, `send(2)`, `write(2V)`, `getprotoent(3N)`, `inet(4F)`, `ip(4P)`, `routing(4N)`

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1064-01)

**BUGS**

Replies to ICMP “echo” messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.



## NAME

ie – Intel 10 Mb/s Ethernet interface

## CONFIG — SUN-4 SYSTEM

device ie0 at obio ? csr 0x6000000 priority 3  
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

## CONFIG — SUN-3x SYSTEM

device ie0 at obio ? csr 0x65000000 priority 3  
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

## CONFIG — SUN-3 SYSTEM

device ie0 at obio ? csr 0xc0000 priority 3  
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75  
device ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74

## CONFIG — SUN-2 SYSTEM

device ie0 at obio 2 csr 0x7f0800 priority 3  
device ie1 at vme24 ? csr 0xe88000 priority 3 vector ieintr 0x75  
device ie0 at mbmem ? csr 0x88000 priority 3  
device ie1 at mbmem ? csr 0x8c000 flags 2 priority 3

## CONFIG — SUN386i SYSTEM

device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3

## DESCRIPTION

The ie interface provides access to a 10 Mb/s Ethernet network through the Intel 82586 controller chip. For a general description of network interfaces see if(4N).

The first Sun-4 and Sun-3x lines above specify CPU-board-resident Intel Ethernet interfaces; the second Sun-4 and Sun-3x lines specify Multibus Intel Ethernet interfaces for use with VME adapters.

In the Sun-3 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interface for use with a VME adapter. The third line specifies the Intel Ethernet interface present on a Sun-3 Eurocard board.

In the Sun-2 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface on a Sun-2/50 or Sun-2/160 system. The second line specifies a Multibus Intel Ethernet controller for use with a VME adapter on these systems. The third line specifies the first Multibus Intel Ethernet controller for a Sun-2/120 or Sun-2/170 system. The fourth line specifies the second such controller for these systems.

The Sun386i line above specifies the CPU-board-resident Intel Ethernet interface.

## SEE ALSO

if(4N)

## DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

**ie%d: Ethernet jammed**

Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

**ie%d: no carrier**

The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

**ie%d: lost interrupt: resetting**

The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

**ie%d: iebark reset**

The 82586 failed to complete a watchdog timeout command in the allotted time. The driver recovers by resetting itself and the chip.

**ie%d: WARNING: requeueing**

The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

**ie%d: panic: scb overwritten**

The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

**ie%d: giant packet**

Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of this message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.



## NAME

if – general properties of network interfaces

## DESCRIPTION

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, lo(4), do not.

At boot time, each interface with underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages can be routed through it. Most interfaces require some part of their address specified with an SIOCSIFADDR IOCTL before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets using arp(4P)), the entire address specified in the ioctl is used.

The following ioctl calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an ifreq structure as its parameter. This structure has the form

```

struct ifreq {
    char    ifr_name[16];          /* name of interface (e.g. "ec0") */
    union {
        struct sockaddr ifru_addr;
        struct sockaddr ifru_dstaddr;
        short  ifru_flags;
    } ifr_ifru;
#define ifr_addr      ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_flags     ifr_ifru.ifru_flags     /* flags */
};

```

|                |                                                                                                                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SIOCSIFADDR    | Set interface address. Following the address assignment, the "initialization" routine for the interface is called.                                                                                                                                                                      |
| SIOCGIFADDR    | Get interface address.                                                                                                                                                                                                                                                                  |
| SIOCSIFDSTADDR | Set point to point address for interface.                                                                                                                                                                                                                                               |
| SIOCGIFDSTADDR | Get point to point address for interface.                                                                                                                                                                                                                                               |
| SIOCSIFFLAGS   | Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.                                                                                                                                                 |
| SIOCGIFFLAGS   | Get interface flags.                                                                                                                                                                                                                                                                    |
| SIOCGIFCONF    | Get interface configuration list. This request takes an ifconf structure (see below) as a value-result parameter. The ifc_len field should be initially set to the size of the buffer pointed to by ifc_buf. On return it will contain the length, in bytes, of the configuration list. |

```

/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct ifconf {
    int    ifc_len;          /* size of associated buffer */
    union {
        caddr_t ifcu_buf;
        struct ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};

```

**SIOCADDMULTI** Enable a multicast address for the interface. A maximum of 64 multicast addresses may be enabled for any given interface.

**SIOCDELMULTI** Disable a previously set multicast address.

**SIOCSPROMISC** Toggle promiscuous mode.

#### SEE ALSO

arp(4P), ec(4S), lo(4)





**NAME**

lofs – loopback virtual file system

**CONFIG**

options LOFS

**SYNOPSIS**

```
#include <sys/mount.h>
mount(MOUNT_LOFS, virtual, flags, dir);
```

**DESCRIPTION**

The loopback file system device allows new, virtual file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original file system, however, *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

*virtual* is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *flags* is either 0 or **M\_RDONLY**. The **M\_RDONLY** flag forces all accesses in the new name space to be read-only; without it, accesses are the same as for the underlying file system. All other **mount(2)** flags are preserved from the underlying file systems.

A loopback mount of '/' onto **/tmp/newroot** allows the entire file system hierarchy to appear as if it were duplicated under **/tmp/newroot**, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to '/', or from a pathname relative to **/tmp/newroot** until such time as a file system is mounted in **/tmp/newroot**, or any of its subdirectories.

Loopback mounts of '/' can be performed in conjunction with the **chroot(2)** system call, to provide a complete virtual file system to a process or family of processes.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another file system hierarchy; rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

The standard RC files perform first **4.2** mounts, then **nfs** mounts, during booting. On Sun386i systems, **lo** (loopback) mounts are performed just after **4.2** mounts. **/etc/fstab** files depending on alternate mount orders at boot time will fail to work as expected. Manual modification of **/etc/rc.local** will be needed to make such mount orders work.

**WARNINGS**

Loopback mounts must be used with care; the potential for confusing users and applications is enormous. A loopback mount entry in **/etc/fstab** must be placed after the mount points of both directories it depends on. This is most easily accomplished by making the loopback mount entry the last in **/etc/fstab**, though see **mount(8)** for further warnings.

**SEE ALSO**

**chroot(2)**, **mount(2)**, **fstab(5)**, **mount(8)**

**BUGS**

Because only directories can be mounted or mounted on, the structure of a virtual file system can only be modified at directories.



## NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

## CONFIG — SUN-3 SYSTEM

## MCP

```
device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88
```

## ALM-2

pseudo-device mcpa64

## SYNOPSIS

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/ttyxy", mode);
open("/dev/ttydn", mode);
open("/dev/cuan", mode);
```

## DESCRIPTION (MCP)

The Sun MCP (Multiprotocol Communications Processor) supports up to four synchronous serial lines in conjunction with SunLink™ Multiple Communication Protocol products.

## DESCRIPTION (ALM-2)

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each port supports those `termio(4)` device control functions specified by flags in the `c_cflag` word of the `termios` structure and by the `IGNBRK`, `IGNPAR`, `PARMRK`, or `INPCK` flags in the `c_iflag` word of the `termios` structure are performed by the `mcp` driver. All other `termio(4)` functions must be performed by `STREAMS` modules pushed atop the driver; when a device is opened, the `ldterm(4M)` and `ttcompat(4M)` `STREAMS` modules are automatically pushed on top of the stream, providing the standard `termio(4)` interface.

Bit *i* of flags may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying flags `0x0004` in the specification of `mcp0` would treat line `/dev/ttyh2` in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named `/dev/ttyXY`, where *X* represents the physical board as one of the characters *h*, *i*, *j*, or *k*, and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is `/dev/ttyh0`, and the sixteenth line on the third board is `/dev/ttyjf.`)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed `/dev/ttydn`, where *n* is a number indicating which dial-in line it is (so that `/dev/ttyd0` is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named `/dev/cuan`, where *n* is the number of the dial-in line.

The `/dev/cuan` lines are special in that they can be opened even when there is no carrier on the line. Once a `/dev/cuan` line is opened, the corresponding tty line cannot be opened until the `/dev/cuan` line is closed; a blocking open will wait until the `/dev/cuan` line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the `/dev/ttydn` line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding `/dev/cuan` line cannot be opened. This allows a modem to be



attached to e.g. `/dev/ttyd0` (renamed from `/dev/ttyh0`) and used for dialin (by enabling the line for login in `/etc/ttytab`) and also used for dialout (by `tip(1C)` or `uucp(1C)`) as `/dev/cua0` when no one is logged in on the line. Note: the bit in the `flags` word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

#### IOCTLS

The standard set of `termio ioctl()` calls are supported by the ALM-2.

If the `CRTSCTS` flag in the `c_cflag` is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the `CRTSCTS` flag is clear, the state of CTS has no effect. Breaks can be generated by the `TCSBRK`, `TIOCSBRK`, and `TIOCCBRK` `ioctl()` calls. The modem control lines `TIOCM_CAR`, `TIOCM_CTS`, `TIOCM_RTS`, and `TIOCM_DTR` are provided.

The input and output line speeds may be set to any of the speeds supported by `termio`. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

#### ERRORS

An `open()` on a `/dev/tty*` or a `/dev/cu*` device will fail if:

|                    |                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ENXIO</code> | The unit being opened does not exist.                                                                                                                                           |
| <code>EBUSY</code> | The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open. |
| <code>EBUSY</code> | The unit has been marked as exclusive-use by another process with a <code>TIOCEXCL ioctl()</code> call.                                                                         |
| <code>EINTR</code> | The open was interrupted by the delivery of a signal.                                                                                                                           |

#### DESCRIPTION (PRINTER PORT)

The printer port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

Minor device numbers in the range 64 – 67 access the printer port, and the recommended naming is `/dev/mcpp[0-3]`.

#### IOCTLS

Various control flags and status bits may be fetched and set on an MCP printer port. The following flags and status bits are supported; they are defined in `sundev/mcpcmd.h`:

|                          |      |                                                              |
|--------------------------|------|--------------------------------------------------------------|
| <code>MCPRIGNSLCT</code> | 0x02 | set if interface ignoring <code>SLCT</code> — on open        |
| <code>MCPRDIAG</code>    | 0x04 | set if printer is in self-test mode                          |
| <code>MCPRVMEINT</code>  | 0x08 | set if VME bus interrupts enabled                            |
| <code>MCPRINTPE</code>   | 0x10 | print message when out of paper                              |
| <code>MCPRINTSLCT</code> | 0x20 | print message when printer offline                           |
| <code>MCPRPE</code>      | 0x40 | set if device ready, cleared if device out of paper          |
| <code>MCPRSLCT</code>    | 0x80 | set if device online (Centronics <code>SLCT</code> asserted) |

The flags `MCPRINTSLCT`, `MCPRINTPE`, and `MCPRDIAG` may be changed; the other bits are status bits and may not be changed.

The `ioctl()` calls supported by MCP printer ports are listed below.

|                       |                                                                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>MCPIOGPR</code> | The argument is a pointer to an <b>unsigned char</b> . The printer flags and status bits are stored in the <b>unsigned char</b> pointed to by the argument. |
| <code>MCPIOSPR</code> | The argument is a pointer to an <b>unsigned char</b> . The printer flags are set from the <b>unsigned char</b> pointed to by the argument.                  |

#### ERRORS

Normally, the interface only reports the status of the device when attempting an `open(2V)` call. An `open()` on a `/dev/mcpp*` device will fail if:



**ENXIO**            The unit being opened does not exist.

**EIO**             The device is offline or out of paper.

Bit 17 of the configuration flags may be specified to say that the interface should ignore Centronics SLCT- and RDY/PE- when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT- and RDY lines are not asserted during an actual data transfer (as with a write(2V) call), no data is transferred.

#### FILES

|                       |                       |
|-----------------------|-----------------------|
| /dev/mcpp[0-3]        | parallel printer port |
| /dev/tty[h-k][0-9a-f] | hardwired tty lines   |
| /dev/ttyd[0-9a-f]     | dialin tty lines      |
| /dev/cua[0-9a-f]      | dialout tty lines     |

#### SEE ALSO

**tip(1C), uucp(1C), mti(4S), termio(4), ldterm(4M), ttcompat(4M), zs(4S)**

#### DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

**mcpn: silo overflow.**

More than *n* characters (*n* very large) have been received by the mcp hardware without being read by the software.

**\*\*\*port *n* supports RS449 interface\*\*\***

Probably an incorrect jumper configuration. Consult the hardware manual.

**mcp port *n* receive buffer error**

The mcp encountered an error concerning the synchronous receive buffer.

**Printer on mcpp*n* is out of paper**

**Printer on mcpp*n* paper ok**

**Printer on mcpp*n* is offline**

**Printer on mcpp*n* online**

Assorted printer diagnostics, if enabled as discussed above.

#### BUGS

Note: pin 4 is used for hardware flow control on ALM-2 ports 0 through 3. These two pins should *not* be tied together on the ALM end.





**NAME**

mti – Systech MTI-800/1600 multi-terminal interface

**CONFIG — SUN-3 SYSTEM**

device mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88  
 device mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89  
 device mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a  
 device mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

**CONFIG — SUN-2 SYSTEM**

device mti0 at mbio ? csr 0x620 flags 0xffff priority 4  
 device mti1 at mbio ? csr 0x640 flags 0xffff priority 4  
 device mti2 at mbio ? csr 0x660 flags 0xffff priority 4  
 device mti3 at mbio ? csr 0x680 flags 0xffff priority 4  
 device mti0 at vme16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88  
 device mti1 at vme16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89  
 device mti2 at vme16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a  
 device mti3 at vme16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

**SYNOPSIS**

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/ttyxy", mode);
open("/dev/ttydn", mode);
open("/dev/cuan", mode);
```

**DESCRIPTION**

The Systech MTI card provides 8 (MTI-800) or 16 (MTI-1600) serial communication lines with modem control. Each port supports those `termio(4)` device control functions specified by flags in the `c_cflag` word of the `termios` structure and by the `IGNBRK`, `IGNPAR`, `PARMRK`, or `INPCK` flags in the `c_iflag` word of the `termios` structure are performed by the `mti` driver. All other `termio(4)` functions must be performed by `STREAMS` modules pushed atop the driver; when a device is opened, the `ldterm(4M)` and `ttcompat(4M)` `STREAMS` modules are automatically pushed on top of the stream, providing the standard `termio(4)` interface.

Bit *i* of `flags` may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying `flags 0x0004` in the specification of `mti0` would treat line `/dev/tty02` in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named `/dev/ttyXY`, where *X* is the physical board number (0 – 3), and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is `/dev/tty00`, and the sixteenth line on the third board is `/dev/tty2f`.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed `/dev/ttydn`, where *n* is a number indicating which dial-in line it is (so that `/dev/ttyd0` is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named `/dev/cuan`, where *n* is the number of the dial-in line.

The `/dev/cuan` lines are special in that they can be opened even when there is no carrier on the line. Once a `/dev/cuan` line is opened, the corresponding tty line can not be opened until the `/dev/cuan` line is closed; a blocking open will wait until the `/dev/cuan` line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the `/dev/ttydn` line has been opened successfully (usually only when carrier is



recognized on the modem) the corresponding `/dev/cuan` line can not be opened. This allows a modem to be attached to e.g. `/dev/ttyd0` (renamed from `/dev/tty00`) and used for dialin (by enabling the line for login in `/etc/ttytab`) and also used for dialout (by `tip(1C)` or `uucp(1C)`) as `/dev/cua0` when no one is logged in on the line. Note: the bit in the `flags` word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

#### WIRING

The Systech requires the CTS modem control signal to operate. If the device does not supply CTS then RTS should be jumpered to CTS at the distribution panel (short pins 4 to 5). Also, the CD (carrier detect) line does not work properly. When connecting a modem, the modem's CD line should be wired to DSR, which the software will treat as carrier detect.

#### IOCTLS

The standard set of `termio ioctl()` calls are supported by `mti`.

The state of the CRTSCTS flag in the `c_cflag` word has no effect; no output will be generated unless CTS is high. Breaks can be generated by the `TCSBRK`, `TIOCSBRK`, and `TIOCCBRK ioctl()` calls. The modem control lines `TIOCM_CAR`, `TIOCM_CTS`, `TIOCM_RTS`, and `TIOCM_DTR` are provided; however, as described above, the DSR line is treated as CD and the CD line is ignored.

The input and output line speeds may be set to any of the speeds supported by `termio`. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed. The baud rates **B200** and **B38400** are not supported by the hardware; **B200** selects 2000 baud, and **B38400** selects 7200 baud.

#### ERRORS

An `open()` will fail if:

|       |                                                                                                                                                                                 |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENXIO | The unit being opened does not exist.                                                                                                                                           |
| EBUSY | The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open. |
| EBUSY | The unit has been marked as exclusive-use by another process with a <code>TIOCEXCL ioctl()</code> call.                                                                         |
| EINTR | The open was interrupted by the delivery of a signal.                                                                                                                           |

#### FILES

|                                    |                     |
|------------------------------------|---------------------|
| <code>/dev/tty[0-3][0-9a-f]</code> | hardwired tty lines |
| <code>/dev/ttyd[0-9a-f]</code>     | dialin tty lines    |
| <code>/dev/cua[0-9a-f]</code>      | dialout tty lines   |

#### SEE ALSO

`tip(1C)`, `uucp(1C)`, `mcp(4S)`, `termio(4)`, `ldterm(4M)`, `ttcompat(4M)`, `zs(4S)`

#### DIAGNOSTICS

Most of these diagnostics "should never happen" and their occurrence usually indicates problems elsewhere in the system.

**mtin, n: silo overflow.**

More than 512 characters have been received by the `mti` hardware without being read by the software. Extremely unlikely to occur.

**mtin: read error code <n>. Probable hardware fault**

The `mti` returned the indicated error code. See the MTI manual.

**mtin: DMA output error.**

The `mti` encountered an error while trying to do DMA output.

**mtin: impossible response n.**

The `mti` returned an error it could not understand.



**NAME**

mtio – general magnetic tape interface

**SYNOPSIS**

```
#include <sys/ioctl.h>
#include <sys/mtio.h>
```

**DESCRIPTION**

Both 1/2" and 1/4" magnetic tape drives share the same general interface regardless of the hardware involved. The remainder of this section discusses the common features of that interface.

The "cooked" magnetic tape device files read and write magnetic tape in 2048 byte blocks (the 2048 is actually `BLKDEV_IOSIZE` in `/usr/include/sys/param.h`). The name of such a device file might be `/dev/mt0`. The final component of the device name is the type of device the file refers to, and the unit number of that device.

These files are rewound when closed, except for "no-rewind" versions. The names of no-rewind device files use the letter `n` as the beginning of the final component. The no-rewind version of `/dev/mt0` would be `/dev/nmt0`. When a 1/2" tape file, opened for writing or just written, is closed, two tape marks are written. If the tape is not to be rewound, it is positioned with the head between the two tapemarks.

The files discussed above are useful for making taped files consistent with ordinary files. This interface requires that all blocks be 2048 bytes long, and does not permit special operations (such as spacing the tape forward or backward) to be performed. The "raw" interface is appropriate when reading or writing long records or using foreign tapes. Raw device files are indicated by the letter `r` before the device type in the device name: the raw version of `/dev/mt0` would be `/dev/rmt0`, and the raw version of `/dev/nmt0` would be `/dev/nrmt0`.

`read(2V)` or `write(2V)` calls read or write the next record on the tape. When the call is to write, the record has the same length as the given buffer. During a read call, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size. In raw tape I/O, seeks are ignored (except `st`). When a tape mark is read, a zero byte count is returned; another read will fetch the first record of the next tape file. Two successive reads returning zero byte counts indicate the end of recorded media.

**1/2" Reel Tape**

Data bytes are recorded in parallel onto the 9-track tape. The number of bytes in a physical record varies between 1 to 65535 bytes. Files end with one file mark except for the last, which signals the end of recorded media with two file marks. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

The recording formats available (check specific tape drive) are 800 BPI, 1600 BPI, and 6250 BPI, and data compression. Actual storage capacity is a function of the recording format and the length of the tape reel. For example, using a 2400 foot tape, 20 MB can be stored using 800 BPI, 40 MB using 1600 BPI, 140 MB using 6250 BPI, or up to 700 MB using data compression.

**1/4" Cartridge Tape**

Data is recorded serially onto 1/4-inch cartridge tape. The number of bytes per record is determined by the physical record size of the device. The I/O request size must be a multiple of the physical record size of the device. For QIC-11, QIC-24, and QIC-150 tape drives the block size is 512 bytes.

The records are recorded on tracks in a serpentine motion. As one track is completed, the drive switches to the next and begins writing in the opposite direction, eliminating the wasted motion of rewinding. Each file, including the last, ends with one file mark.

Files may be written only at the beginning of the tape or after the last written file. This prevents corrupting data by overwriting files.

Storage capacity is based on the number of tracks the drive is capable of recording. For example, 4-track drives can only record 20 MB of data on a 450 foot tape; 9-track drives can record up to 45 MB of data on a tape of the same length.. QIC-11 is the only tape format available for 4-track tape drives. In contrast, 9-track tape drives can use either QIC-24 or QIC-11. Storage capacity is not appreciably affected by using

either format. QIC-24 is preferable to QIC-11 because it records a reference signal to mark the position of the first track on the tape, and each block has a unique block number.

The QIC-150 tape drives require DC-6150 (or equivalent) tape cartridges for writing. However, they can read other tape cartridges in QIC-11, QIC-24, QIC-120, or QIC-150 tape formats.

A number of additional ioctl operations are available on "raw" devices. The following definitions are from `/usr/include/sys/mtio.h`:

```
/*
 * Structures and definitions for mag tape io control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct mtop {
    short    mt_op;        /* operations defined below */
    daddr_t mt_count;      /* how many of them */
};

#define MTWEOF      0      /* write an end-of-file record */
#define MTFSF       1      /* forward space file */
#define MTBSF       2      /* backward space file */
#define MTFSR       3      /* forward space record */
#define MTBSR       4      /* backward space record */
#define MTREW       5      /* rewind */
#define MTOFFL      6      /* rewind and put the drive offline */
#define MTNOP       7      /* no operation, sets status only */
#define MTRETEN     8      /* retension the tape */
#define MTERASE     9      /* erase the entire tape */
#define MTEOM       10     /* position to end of media (SCSI only) */
#define MTBSFM      11     /* backward space file mark */

/* structure for MTIOCGET - mag tape get status command */
struct mtget {
    short    mt_type;      /* type of magtape device */
    /* the following two registers are grossly device dependent */
    short    mt_dsreg;     /* "drive status" register */
    short    mt_erreg;     /* "error" register */
    /* optional error info. */
    daddr_t mt_resid;      /* residual count */
    daddr_t mt_fileno;     /* file number of current position */
    daddr_t mt_blkno;      /* block number of current position */
};

/*
 * Constants for mt_type byte
 */
#define MT_ISTS      0x01  /* vax: unibus ts-11 */
#define MT_ISHT      0x02  /* vax: massbus tu77, etc */
#define MT_ISTM      0x03  /* vax: unibus tm-11 */
#define MT_ISMT      0x04  /* vax: massbus tu78 */
#define MT_ISUT      0x05  /* vax: unibus gcr */
#define MT_ISCPC     0x06  /* sun: Multibus tapemaster */
#define MT_ISAR      0x07  /* sun: Multibus archive */
#define MT_ISSC      0x08  /* sun: SCSI archive */
```



```

#define MT_ISXY          0x09    /* sun: Xylogics 472 */

#define MT_ISSYSGEN11    0x10    /* sun: SCSI Sysgen, QIC-11 only */
#define MT_ISSYSGEN      0x11    /* sun: SCSI Sysgen QIC-24/11 */
#define MT_ISDEFAULT     0x12    /* sun: SCSI default CCS */
#define MT_ISCCS3        0x13    /* sun: SCSI generic (unknown) CCS */
#define MT_ISMT02        0x14    /* sun: SCSI Emulex MT02 */
#define MT_ISVIPER1       0x15    /* sun: SCSI Archive QIC-150 Viper */
#define MT_ISWANGTEK1     0x16    /* sun: SCSI Wangtek QIC-150 */
#define MT_ISCCS7         0x17    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS8         0x18    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS9         0x19    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS11        0x1a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS12        0x1b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS13        0x1c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS14        0x1d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS15        0x1e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS16        0x1f    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCDC          0x20    /* sun: SCSI CDC 1/2" cartridge */
#define MT_ISFUJI         0x21    /* sun: SCSI Fujitsu 1/2" cartridge */
#define MT_ISKENNEDY      0x22    /* sun: SCSI Kennedy 1/2" reel */
#define MT_ISHP           0x23    /* sun: SCSI HP 1/2" reel */
#define MT_ISCCS21        0x24    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS22        0x25    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS23        0x26    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS24        0x27    /* sun: SCSI generic (unknown) CCS */
#define MT_ISEXABYTE      0x28    /* sun: SCSI Exabyte 8mm cartridge */
#define MT_ISCCS26        0x29    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS27        0x2a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS28        0x2b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS29        0x2c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS30        0x2d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS31        0x2e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS32        0x2f    /* sun: SCSI generic (unknown) CCS */

/*
 * Device table structure and data for looking tape name from
 * tape id number. Used by mt.c.
 */
struct mt_tape_info {
    short    t_type;        /* type of magtape device */
    char     *t_name;       /* printing name */
    char     *t_dsbits;     /* "drive status" register */
    char     *t_erbits;     /* "error" register */
};

#define MT_TAPE_INFO { \
    { MT_ISCPC,          "TapeMaster",          TMS_BITS, 0 }, \
    { MT_ISXY,           "Xylogics 472",        XTS_BITS, 0 }, \
    { MT_ISAR,           "Archive",              ARCH_CTRL_BITS, ARCH_BITS }, \
    { MT_ISSYSGEN11,     "Sysgen QIC-11",        0, 0 }, \
    { MT_ISSYSGEN,       "Sysgen",               0, 0 }, \
    { MT_ISMT02,         "Emulex MT-02",        0, 0 }, \

```

```

{ MT_ISVIPER1, "Archive QIC-150",    0, 0 }, \
{ MT_ISWANGTEK1, "Wangtek QIC-150",    0, 0 }, \
{ MT_ISCDC, "CDC",    0, 0 }, \
{ MT_ISKENNEDY, "Kennedy",    0, 0 }, \
{ MT_ISHP, "HP-88780",    0, 0 }, \
{ MT_ISEXABYTE, "Exabyte",    0, 0 }, \
{ 0 } \
}

/*
 * Constants for mt_type byte
 */

/*
 * Check if mt_type is one of the SCSI tape devices.
 */
#define MT_TYPE SCSI(mt_type) \
    ((mt_type >= MT_ISSYSGEN11) && (mt_type <= MT_ISCCS32))

/*
 * Older 1/4-inch cartridge tapes devices.
 * A blocking factor of 126 is recommended for compatibility.
 * A larger blocking factor may be used for improved streaming
 * performance.
 */
#define MT_TYPE_OLD_CARTRIDGE(mt_type) \
    ((mt_type == MT_ISSYSGEN11) || (mt_type == MT_ISSYSGEN) || \
    (mt_type == MT_ISAR))

/*
 * Current 1/4-inch cartridge tape devices.
 * A blocking factor of 40 (to 60) is recommended for
 * optimal streaming performance.
 */
#define MT_TYPE_NEW_CARTRIDGE(mt_type) \
    (mt_type >= MT_ISDEFAULT && mt_type <= MT_ISCCS16)

/*
 * All 1/4-inch cartridge tape devices.
 * A blocking factor of 126 is recommended for compatibility
 * (during writes). See above for specific recommendations for
 * reading.
 */
#define MT_TYPE_CARTRIDGE(mt_type) \
    ((mt_type >= MT_ISSYSGEN11 && mt_type <= MT_ISCCS16) || \
    (mt_type == MT_ISAR))

/*
 * All 1/2-inch reel tape devices.
 * A blocking factor of 20 is recommended for compatibility.
 */
#define MT_TYPE_REEL(mt_type) \

```



```
((mt_type >= MT_ISCDC && mt_type <= MT_ISCCS32) || \
(mt_type >= MT_ISTS && mt_type <= MT_ISCPC) || \
(mt_type == MT_ISXY))
```

```
/* mag tape io control commands */
```

```
#define MTIOCTOP    _IOW(m, 1, struct mtop) /* do a mag tape op */
```

```
#define MTIOCGET    _IOR(m, 2, struct mtget) /* get tape status */
```

```
#ifndef KERNEL
```

```
#define DEFTAPE     "/dev/rmt12"
```

```
#endif
```

#### SEE ALSO

mt(1), tar(1), read(2V), write(2V), ar(4S), st(4S), tm(4S), xt(4S)

#### BUGS

"Cooked" mode does not work for all magnetic tape devices (st, tm, and xt for example).

**NAME**

sd – driver for SCSI disk devices

**CONFIG — SUN-3/4 SYSTEMS**

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40

controller si0 at obio ? csr 0x140000 priority 2

disk sd0 at si0 drive 0 flags 0

disk sd1 at si0 drive 1 flags 0

disk sd2 at si0 drive 8 flags 0

disk sd3 at si0 drive 9 flags 0

disk sd4 at si0 drive 16 flags 0

disk sd6 at si0 drive 24 flags 0

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40

disk sd0 at sc0 drive 0 flags 0

disk sd1 at sc0 drive 1 flags 0

disk sd2 at sc0 drive 8 flags 0

disk sd3 at sc0 drive 9 flags 0

disk sd4 at sc0 drive 16 flags 0

disk sd6 at sc0 drive 24 flags 0

The first two controller lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third controller line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

The four lines following the controller specification lines define the available disk devices, sd0 – sd6.

The flags field is used to specify the SCSI device type to the host adapter. flags must be set to 0 to identify disk devices.

The drive value is calculated using the formula:

$$8 * target + lun$$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following si0 and si1 above, describes the configuration for the older sc0 host adapter. It uses the same configuration description as the si0 host adapter.

**CONFIG — SPARCsystem 330 and SUN-3/80 SYSTEMS**

controller sm0 at obio ? csr 0xfa000000 priority 2

disk sd0 at sm0 drive 0 flags 0

disk sd1 at sm0 drive 1 flags 0

disk sd2 at sm0 drive 8 flags 0

disk sd3 at sm0 drive 9 flags 0

disk sd4 at sm0 drive 16 flags 0

disk sd6 at sm0 drive 24 flags 0

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, sm0. It follows the same rules as described above for the Sun-3, Sun-3x, Sun-4 section.

**CONFIG — SUN-4/110 SYSTEM**

controller sw0 at obio 2 csr 0xa000000 priority 2

disk sd0 at sw0 drive 0 flags 0

disk sd1 at sw0 drive 1 flags 0

disk sd2 at sw0 drive 8 flags 0

disk sd3 at sw0 drive 9 flags 0

disk sd4 at sw0 drive 16 flags 0

disk sd6 at sw0 drive 24 flags 0



The Sun-4/110 uses an on-board SCSI host adapter, sw0. It follows the same rules as described above for the Sun-3, Sun-3x, Sun-4 section.

#### CONFIG — SUN-2 SYSTEM

```
controller sc0 at mbmem ? csr 0x80000 priority 2
controller sc1 at mbmem ? csr 0x84000 priority 2
controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd2 at sc1 drive 0 flags 0
disk sd3 at sc1 drive 1 flags 0
```

The controller lines above specify the first and second SCSI host adapters, sc0 and sc1, on Sun-2/120 and Sun-2/170 systems. The third controller line specifies the first and only host adapter on a Sun-2/160 system. It follows the same rules as described under the Sun-3, Sun-3x, Sun-4 configuration section above.

#### CONFIG — SUN-3/E SYSTEM

```
controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
disk sd0 at se0 drive 0 flags 0
disk sd1 at se0 drive 1 flags 0
disk sd2 at se0 drive 8 flags 0
disk sd3 at se0 drive 9 flags 0
```

The Sun-3/E uses a VME-based SCSI host adapter, se0. It follows the same rules as described above for the Sun-3, Sun-3x, Sun-4 section.

#### CONFIG — Sun386i

```
controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
disk sd0 at wds0 drive 0 flags 0
disk sd1 at wds0 drive 8 flags 0
disk sd2 at wds0 drive 16 flags 0
```

The Sun-386i configuration follows the same rules described above under the Sun-3, Sun-3x, Sun-4 configuration section.

#### DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0. The standard device names begin with "sd" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-6.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the raw files conventionally begin with an extra 'r.'

I/O requests (for example lseek(2)) to the SCSI disk must have an offset that is a multiple of 512 bytes (DEV\_BSIZE), or the driver returns an EINVAL error. If the transfer length is not a multiple of 512 bytes, the transfer count is rounded up by the driver.

#### Disk Support

This driver handles the Adaptec ACB-4000 disk controller for ST-506 drives, the Emulex MD21 disk controller for ESDI drives, and embedded, CCS-compatible SCSI disk drives.

On Sun386i systems, this driver supports the CDC Wren III half-height, and Wren IV full-height SCSI disk drives.

The type of disk drive is determined using the SCSI inquiry command and reading the volume label stored on block 0 of the drive. The volume label describes the disk geometry and partitioning; it must be present or the disk cannot be mounted by the system.

The **sd?a** partition is normally used for the root file system on a disk, the **sd?b** partition as a paging area (e.g. swap), and the **sd?c** partition for pack-pack copying. **sd?c** normally maps the entire disk and may also be used as the mount point for secondary disks in the system. The rest of the disk is normally the **sd?g** partition. For the primary disk, the user file system is located here.

**FILES**

**/dev/sd[0-6][a-h]**      block files  
**/dev/rsd[0-6][a-h]**      raw files

**SEE ALSO**

**dkio(4S), directory(3), lseek(2), read(2V), write(2V)**  
 Product Specification for Wren IV SCSI Model 94171  
 Product Specification for Wren III SCSI Model 94161  
 Product Specification for Wren III SCSI Model 94211  
 Emulex MD21 Disk Controller Programmer Reference Manual  
 Adaptec ACB-4000 Disk Controller OEM Manual

**DIAGNOSTICS****sd?: sdtimer: I/O request timeout**

A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

**sd?: sdtimer: can't abort request**

The driver is unable to find the request in the disconnect queue to notify the device driver that it has failed.

**sd?: no space for inquiry data****sd?: no space for disk label**

The driver was unable to get enough space for temporary storage. The driver is unable to open the disk device.

**sd?: <%s>**

The driver has found a SCSI disk device and opened it for the first time. The disk label is displayed to notify the user.

**sd?: SCSI bus failure**

A host adapter error was detected. The system may need to be rebooted.

**sd?: single sector I/O failed**

The driver attempted to recover from a transfer by writing each sector, one at a time, and failed. The disk needs to be reformatted to map out the new defect causing this error.

**sd?: retry failed****sd?: rezero failed**

A disk operation failed. The driver first tries to recover by retrying the command, if that fails, the driver rezeros the heads to cylinder 0 and repeats the retries. A failure of either the retry or rezero operations results in these warning messages; the error recovery operation continues until the retry count is exhausted. At that time a hard error is posted.

**sd?: request sense failed**

The driver was attempting to determine the cause of an I/O failure and was unable to get more information. This implies that the disk device may have failed.

**sd?: warning, abs. block %d has failed %d times**

The driver is warning the user that the specified block has failed repeatedly.

**sd?: block %d needs mapping****sd?: reassigning defective abs. block %d**

The specified block has failed repeatedly and may soon become an unrecoverable failure. If the driver does not map out the specified block automatically, it is recommended that the user



correct the problem.

**sd?: reassign block failed**

The driver attempted to map out a block having excessive soft errors and failed. The user needs to run format and repair the disk.

**sd?%c: cmd how blk %d (rel. blk %d)**

**sense key(0x%x): %s, error code(0x%x): %s**

An I/O operation (**cmd**), encountered an error condition at absolute block (**blk %d**), partition (**sd?%c:**), or relative block (**rel. block %d**). The error recovery operation (**how**) indicates whether it *retry'ed*, *restored*, or *failed*. The **sense key** and **error code** of the error are displayed for diagnostic purposes. The absolute **blk** of the the error is used for mapping out the defective block. The **rel. blk** is the block (sector) in error, relative to the beginning of the partition involved. This is useful for using **icheck(8)** to repair a damaged file structure on the disk.

**NAME**

sockio – ioctls that operate directly on sockets

**SYNOPSIS**

```
#include <sys/sockio.h>
```

**DESCRIPTION**

The IOCTL's listed in this manual page apply directly to sockets, independent of any underlying protocol. Note: the **setsockopt** system call (see **getsockopt(2)**) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. **ioctls** for a specific network interface or protocol are documented in the manual page for that interface or protocol.

|                    |                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SIOCSPGRP</b>   | The argument is a pointer to an <b>int</b> . Set the process-group ID that will subsequently receive <b>SIGIO</b> or <b>SIGURG</b> signals for the socket referred to by the descriptor passed to <b>ioctl</b> to the value of that <b>int</b> .                                    |
| <b>SIOCGPGRP</b>   | The argument is a pointer to an <b>int</b> . Set the value of that <b>int</b> to the process-group ID that is receiving <b>SIGIO</b> or <b>SIGURG</b> signals for the socket referred to by the descriptor passed to <b>ioctl</b> .                                                 |
| <b>SIOCCATMARK</b> | The argument is a pointer to an <b>int</b> . Set the value of that <b>int</b> to 1 if the read pointer for the socket referred to by the descriptor passed to <b>ioctl</b> points to a mark in the data stream for an out-of-band message, and to 0 if it does not point to a mark. |

**SEE ALSO**

**ioctl(2)**, **getsockopt(2)**, **filio(4)**



## NAME

st – driver for SCSI tape devices

## CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

```
controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller si0 at obio ? csr 0x140000 priority 2
tape st0 at si0 drive 32 flags 1
tape st1 at si0 drive 40 flags 1
tape st2 at si1 drive 32 flags 1
tape st3 at si1 drive 40 flags 1
```

```
controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1
```

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

Following the **controller** specification lines are four lines which define the available **tape** devices, **st0**–**st3**. The first two **tape** devices, **st0** and **st1**, are on the first **controller**, **si0**. The next two **tape** devices, **st2** and **st3**, are on the second **controller**, **si1**.

The **flags** field is used to specify the SCSI device type to the host adapter. The **flags** field must be set to 1 to identify **tape** devices.

The **drive** value is calculated using the formula:

$$8 * target + lun$$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the older **sc0** host adapter configuration. It follows the same configuration description as the **si0** host adapter.

## CONFIG — SPARCsystem 330, SUN-3/80 SYSTEMS

```
controller sm0 at obio ? csr 0xfa000000 priority 2
tape st0 at sm0 drive 32 flags 1
tape st1 at sm0 drive 40 flags 1
```

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

## CONFIG — SUN-4/110 SYSTEM

```
controller sw0 at obio 2 csr 0xa000000 priority 2
tape st0 at sw0 drive 32 flags 1
tape st1 at sw0 drive 40 flags 1
```

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

## CONFIG — SUN-3/E SYSTEM

```
controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape st0 at se0 drive 32 flags 1
tape st1 at se0 drive 40 flags 1
```

The Sun-3/E uses a VME-based SCSI host adapter, **se0**, which follows the rules described above for Sun-3, Sun-3x, Sun-4 systems.

**CONFIG — Sun386i**

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2  
 tape st0 at wds0 drive 32 flags 1

The Sun386i configuration follows the rules described above in the Sun-3, Sun-3x, Sun-4 configuration section.

**CONFIG — SUN-2 SYSTEM**

controller sc0 at mbmem ? csr 0x80000 priority 2  
 controller sc1 at mbmem ? csr 0x84000 priority 2  
 controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40  
 tape st0 at sc0 drive 32 flags 1  
 tape st1 at sc0 drive 40 flags 1  
 tape st0 at sc1 drive 32 flags 1  
 tape st1 at sc1 drive 40 flags 1

The controller lines above specify the first and second SCSI host adapters, sc0 and sc1, on Sun-2/120 and Sun-2/170 systems. The third controller line specifies the only host adapter on a Sun-2/160. It follows the rules described in the Sun-3, Sun-3x, Sun-4 configuration section above.

**DESCRIPTION**

The st device driver is an interface to various SCSI tape devices. Supported 1/4-inch cartridge devices include the Archive Viper QIC-150 streaming tape drive, the Emulex MT-02 tape controller, and the Sysgen SC4000 tape controller. st provides a standard interface to these various devices, see mtio(4) for details.

The driver can be opened with either rewind on close (/dev/rst\*) or no rewind on close (/dev/nrst\*) options. A maximum of four tape formats per device are supported (see FILES below). The tape format is specified using the device name. The four rewind on close formats for st0, for example, are /dev/rst0, /dev/rst8, /dev/rst16, and /dev/rst24.

**Read Operation**

Fixed-length I/O tape devices require the number of bytes read or written to be a multiple of the physical record size. For example, 1/4-inch cartridge tape devices only read or write multiples of 512 bytes.

Fixed-length tape devices read or write multiple records if the blocking factor is greater than 64512 bytes (minphys limit). These multiple writes are limited to 64512 bytes. For example, if a write request is issued for 65536 bytes using a 1/4-inch cartridge tape, two writes are issued; the first for 64512 bytes and the second for 1024 bytes.

Tape devices, which support variable-length I/O operations, such as 1/2-inch reel tape, may read or write a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver reads or writes multiple records to satisfy the request. These multiple records are limited to 65534 bytes. As an example, if a write request for 65540 bytes is issued using 1/2-inch reel tape, two records are written; one for 65534 bytes followed by one for 6 bytes.

If the driver is opened for reading in a different format than the tape is written in, the driver overrides the user selected format. For example, if a 1/4-inch cartridge tape is written in QIC-24 format and opened for reading in QIC-11, the driver will detect a read failure on the first read and automatically switch to QIC-24 to recover the data.

Note: If the /dev/\*st[0-3] format is used, no indication is given that the driver has overridden the user selected format. Other formats issue a warning message to inform the user of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2-inch reel and QIC-150 tape drives for example).

If a file mark is encountered during reading, no error is reported but the number of bytes transferred is zero. The next read operation reads into the next file.





End of media is indicated by two successive zero transfer counts. No further reading should be performed past the end of recorded media.

If the read request size is 2048 bytes, the tape driver behaves as a disk device and honors seek positioning requests (see `lseek(2)`). If a file mark is crossed during a read operation, this function is disabled.

#### Write Operation

Writing is allowed at either the beginning of tape or after the last written file on the tape. Writing from the beginning of tape is performed in the user-specified format. The original tape format is used for appending onto previously written tapes. A warning message is issued if the driver has to override the user-specified format.

Care should be used when appending files onto 1/2-inch reel tape devices, since an extra file mark is appended after the last file to mark the end of recorded media. In other words, the last file on the tape ends with two file marks instead of one. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the end of recorded media `ioctl` is provided to eliminate this problem by having the driver perform the positioning operation.

If the end of tape is encountered during writing, no error is reported but the number of bytes transferred is zero and no further writing is allowed. Trailer records may be written by first writing a file mark followed by the trailer records. It is important that these trailer records be kept as short as possible to prevent data loss.

#### Close Operation

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device name is used, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning `ioctl`, like `rewind`, is issued after writing, a file mark is written before repositioning the tape.

Note: For 1/2-inch reel tape devices, two file marks are written to mark the end of recorded media before rewinding or performing a file positioning `ioctl`. If the user wrote a file mark before closing a 1/2-inch reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly.

If no data was written and the driver was opened for WRITE-ONLY access, a file mark is written thus creating a null file.

#### IOCTLS

The following `ioctls` are supported: `forwardspace record`, `forwardspace file`, `backspace record`, `backspace file`, `backspace file mark`, `rewind`, `write file mark`, `offline`, `erase`, `retension`, `space to EOM`, and `get status`.

The `backspace file` and `forwardspace file` tape operations are inverses. Thus, a `forwardspace "-1"` file is equivalent to a `backspace "1"` file. A `backspace "0"` file is the same as `forwardspace "0"` file; both position the tape device to the beginning of the current file.

`Backspace file mark` moves the tape backwards by file marks. The tape position will end on the beginning of tape side of the desired file mark. Devices which do not support this function, such as 1/4-inch cartridge tape, return an `ENXIO` error.

`Backspace record` and `forwardspace record` operations perform much like `space file` operations, except that they move by records instead of files. Variable-length I/O devices (1/2-inch reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4-inch cartridge tape, for example, spaces 512 byte physical records. The status `ioctl` residue count contains the number of files or records not skipped. Record skipping does not go past a file mark; file skipping does not go past the end of recorded media.

`Spacing to the end of recorded media` positions the tape at a location just after the last file written on the tape. For 1/4-inch cartridge tape, this is after the last file mark on the tape. For 1/2-inch reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.



The offline ioctl rewinds and, if appropriate, takes the device offline by unloading the tape. Tape must be inserted before the tape device can be used again.

The erase ioctl rewinds the tape, erases it completely, and returns to the beginning of tape.

The retension ioctl only applies to 1/4-inch cartridge tape devices. It is used to restore tape tension improving the tape's soft error rate after extensive start-stop operations or long-term storage. Devices which do not support this function, such as 1/2-inch reel tape, return an ENXIO error.

The get status ioctl call returns the drive id (mt\_type), sense key error (mt\_erreg), file number (mt\_fileno), and record number (mt\_blkno) of the last error. The residue count (mt\_resid) is set to the number of bytes not transferred or files/records not spaced.

Note: The error status is reset by the get status ioctl call or the next read, write, or other ioctl operation. If no error has occurred (sense key is zero), the current file and record position are returned.

## ERRORS

|        |                                                                                                                                                                                                                                                 |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EACCES | The driver is opened for write access and the tape is write protected, or an attempt is made to write on a write protected tape. For writing with QIC-150 tape drives, this error is also reported if the wrong tape media is used for writing. |
| EBUSY  | The tape device is already in use.                                                                                                                                                                                                              |
| EIO    | During opening, the tape device is not ready because either no tape is in the drive, or the drive is not on-line. Once open, this error is returned if the requested I/O transfer could not be completed.                                       |
| EINVAL | The number of bytes read or written is not a multiple of the physical record size (fixed-length tape devices only).                                                                                                                             |
| ENXIO  | During opening, the tape device does not exist. On ioctl functions, this indicates that the tape device does not support the ioctl function.                                                                                                    |

## FILES

For QIC-150 tape devices (Archive Viper):

```

/dev/rst[0-3]    QIC-150 Format
/dev/rst[8-11]   QIC-150 Format
/dev/rst[16-20]  QIC-150 Format
/dev/rst[24-28]  QIC-150 Format
/dev/nrst[0-3]   non-rewinding QIC-150 Format
/dev/nrst[8-11]  non-rewinding QIC-150 Format
/dev/nrst[16-19] non-rewinding QIC-150 Format
/dev/nrst[24-27] non-rewinding QIC-150 Format

```

For QIC-24 tape devices (Emulex MT-02 and Sysgen SC4000):

```

/dev/rst[0-3]    QIC-11 Format
/dev/rst[8-11]   QIC-24 Format
/dev/rst[16-20]  QIC-24 Format
/dev/rst[24-28]  QIC-24 Format
/dev/nrst[0-3]   non-rewinding QIC-11 Format
/dev/nrst[8-11]  non-rewinding QIC-24 Format
/dev/nrst[16-19] non-rewinding QIC-24 Format
/dev/nrst[24-27] non-rewinding QIC-24 Format

```

Note: The QIC-24 format is preferred over QIC-11 for Sun-3, Sun-3x, Sun-4, and Sun386i systems.

## SEE ALSO

**mt(1), tar(1), mtio(4), dump(8), restore(8)**

Archive Viper QIC-150 Tape Drive Product Specification

Emulex MT-02 Intelligent Tape Controller Product Specification

Sysgen SC4000 Intelligent Tape Controller Product Specification

**DIAGNOSTICS****st?: sttimer: I/O request timeout**

A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

**st?: sttimer: can't abort request**

The driver is unable to find the request in the disconnect que to notify the device driver that it has failed. A SCSI bus reset is issued to recover from this error.

**st?: unknown SCSI device found**

The SCSI device is not a tape device; it is some other type of SCSI device.

**st?: warning, unknown tape drive found**

The driver does not recognize the tape device. Only the default tape density is used; block size is set to the value specified by the tape drive.

**st?: tape is write protected**

The tape is write protected.

**st?: wrong tape media for writing**

For QIC-150 tape drives, this indicates that the user is trying to write on a DC-300XL (or equivalent) tape. Only DC-6150 (or equivalent) tapes can be used for writing.

Note: DC-6150 was formerly known as DC-600XTD.

**st?: warning, rewinding tape**

The driver is rewinding tape in order to set the tape format.

**st?: warning, using alternate tape format**

The driver is overriding the user-selected tape format and using the previously used format.

**st?: warning, tape rewind**

For Sysgen tape controllers, the tape may be rewound as a result of getting sense data.

**st?: format change failed**

The tape drive rejected the mode select command to change the tape format.

**st?: file mark write failed**

The driver was unable to write a file mark.

**st?: warning, The tape may be wearing out or the head may need cleaning.****st?: read retries= %d, file= %d, block= %d****st?: write retries= %d, file= %d, block= %d**

The number of allowable soft errors has been exceeded for this tape. Either the tape heads need cleaning or the tape is wearing out. If the tape is wearing out, continued usage of it is not recommended.

**st?: illegal command**

The SCSI command just issued was illegal. This message can result from issuing an inappropriate command, such as trying to write over previously written files on the tape. On foreign tape devices, this can also be caused by selecting the wrong tape format.

**st?: error: sense key(0x%x): %s, error code(0x%x): %s**

An error has occurred. The sense key message and error code are displayed for diagnostic purposes.

**st?: stread: not modulo %d block size****st?: stwrite: not modulo %d block size**

The read or write request size must be a multiple of the %d physical block size.

**st?: file positioning error****st?: block positioning error**

The driver was unable to position the tape to the desired file or block (record). This is probably caused by a damaged tape.



**BUGS**

Foreign tape devices which do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for Sun supplied tape devices.

Foreign tape devices which do not report a blank check error at the end of recorded media cause file positioning operations to fail. Some tape drives for example, mistakenly report media error instead of blank check error.

"Cooked" mode for read and write operations is not supported.

Systems using the older `sc0` host adapter or the Sysgen SC4000 tape controller, prevent disk I/O over the SCSI bus while the tape is in use (during a rewind for example). This problem is caused by the fact that they do not support disconnect/reconnect to free the SCSI bus. Newer tape devices, like the the Emulex MT-02, and host adapters, like `si0`, eliminate this problem.

Some older systems may not support the QIC-24 format, and may complain (or exhibit erratic behavior) when the user attempts to use this format.





**NAME**

streamio – STREAMS ioctl commands

**SYNOPSIS**

```
#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;
```

**DESCRIPTION**

STREAMS (see `intro(2)`) ioctl commands are a subset of `ioctl(2)` commands that perform a variety of control functions on STREAMS. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *streamhead*. Certain combinations of these arguments may be passed to a module or driver in the stream.

*fildes* is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. Subsequent system calls will fail with *errno* set to this value.

**IOCTLS**

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

|               |                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>I_PUSH</b> | Pushes the module whose name is pointed to by <i>arg</i> onto the top of the current stream, just below the <i>streamhead</i> . It then calls the open routine of the newly-pushed module.                                                                                                                                      |
|               | I_PUSH will fail if one of the following occurs:                                                                                                                                                                                                                                                                                |
|               | EINVAL                      The module name is invalid.                                                                                                                                                                                                                                                                         |
|               | EFAULT <i>arg</i> points outside the allocated address space.                                                                                                                                                                                                                                                                   |
|               | ENXIO                      The open routine of the new module failed.                                                                                                                                                                                                                                                           |
|               | ENXIO                      A hangup is received on the stream referred to by <i>fildes</i> .                                                                                                                                                                                                                                    |
| <b>I_POP</b>  | Removes the module just below the <i>stream head</i> of the stream pointed to by <i>fildes</i> . <i>arg</i> should be 0 in an I_POP request.                                                                                                                                                                                    |
|               | I_POP will fail if one of the following occurs:                                                                                                                                                                                                                                                                                 |
|               | EINVAL                     No module is present on <i>stream</i> .                                                                                                                                                                                                                                                              |
|               | ENXIO                      A hangup is received on the stream referred to by <i>fildes</i> .                                                                                                                                                                                                                                    |
| <b>I_LOOK</b> | Retrieves the name of the module just below the <i>stream head</i> of the stream pointed to by <i>fildes</i> , and places it in a NULL terminated character string pointed at by <i>arg</i> . The buffer pointed to by <i>arg</i> should be at least FMNAMESZ+1 bytes long. An '#include <sys/conf.h>' declaration is required. |
|               | I_LOOK will fail if one of the following occurs:                                                                                                                                                                                                                                                                                |
|               | EFAULT <i>arg</i> points outside the allocated address space of the process.                                                                                                                                                                                                                                                    |
|               | EINVAL                     No module is present on <i>stream</i> .                                                                                                                                                                                                                                                              |

**I\_FLUSH**

This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

|                |                              |
|----------------|------------------------------|
| <b>FLUSHR</b>  | Flush read queues.           |
| <b>FLUSHW</b>  | Flush write queues.          |
| <b>FLUSHRW</b> | Flush read and write queues. |

**I\_FLUSH** will fail if one of the following occurs:

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <b>EAGAIN</b> | No buffers could be allocated for the flush message.            |
| <b>EINVAL</b> | The value of <i>arg</i> is invalid.                             |
| <b>ENXIO</b>  | A hangup is received on the stream referred to by <i>fdes</i> . |

**I\_SETSIG**

Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal (see `sigvec(2)`) when a particular event has occurred on the stream associated with *fdes*. **I\_SETSIG** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

|                 |                                                                                                                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>S_INPUT</b>  | A non-priority message has arrived on a <i>stream head</i> read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length. |
| <b>S_HIPRI</b>  | A priority message is present on the <i>stream head</i> read queue. This is set even if the message is of zero length.                                                                                      |
| <b>S_OUTPUT</b> | The write queue just below the <i>stream head</i> is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.                                       |
| <b>S_MSG</b>    | A STREAMS signal message that contains the SIGPOLL signal has reached the front of the <i>stream head</i> read queue.                                                                                       |

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S\_HIPRI**.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using **I\_SETSIG**. If several processes register to receive this signal for the same event on the same *stream*, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals.

**I\_SETSIG** will fail if one of the following occurs:

|               |                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>EINVAL</b> | The value of <i>arg</i> is invalid or <i>arg</i> is zero and the process is not registered to receive the SIGPOLL signal. |
| <b>EAGAIN</b> | A data structure could not be allocated to store the signal request.                                                      |

**I\_GETSIG**

Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I\_SETSIG** above.



**I\_GETSIG** will fail if one of the following occurs:

- |               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <b>EINVAL</b> | The process is not registered to receive the SIGPOLL signal.          |
| <b>EFAULT</b> | <i>arg</i> points outside the allocated address space of the process. |

### **I\_FIND**

This request compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present.

**I\_FIND** will fail if one of the following occurs:

- |               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <b>EFAULT</b> | <i>arg</i> points outside the allocated address space of the process. |
| <b>EINVAL</b> | <i>arg</i> does not point to a valid module name.                     |

### **I\_PEEK**

This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```

    struct strbuf  ctlbuf;
    struct strbuf  databuf;
    long          flags;

```

The *maxlen* field in the *ctlbuf* and *databuf* *strbuf* structures (see *getmsg(2)*) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to **RS\_HIPRI**, **I\_PEEK** will only look for a priority message on the *stream head* read queue.

**I\_PEEK** returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the **RS\_HIPRI** flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or **RS\_HIPRI**.

**I\_PEEK** will fail if one of the following occurs:

- |               |                                                                                                                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EFAULT</b> | <i>arg</i> points, or the buffer area specified in <i>ctlbuf</i> or <i>databuf</i> is, outside the allocated address space of the process. |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|

### **I\_SRDOPT**

Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

- |              |                                |
|--------------|--------------------------------|
| <b>RNORM</b> | Byte-stream mode, the default. |
| <b>RMSGD</b> | Message-discard mode.          |
| <b>RMSGN</b> | Message-nondiscard mode.       |

Read modes are described in *read(2V)*.

**I\_SRDOPT** will fail if one of the following occurs:

- |               |                                                  |
|---------------|--------------------------------------------------|
| <b>EINVAL</b> | <i>arg</i> is not one of the above legal values. |
|---------------|--------------------------------------------------|

### **I\_GRDOPT**

Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read(2V)*.

**I\_GRDOPT** will fail if one of the following occurs:

- |               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <b>EFAULT</b> | <i>arg</i> points outside the allocated address space of the process. |
|---------------|-----------------------------------------------------------------------|

**I\_NREAD**

Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue.

**I\_NREAD** will fail if one of the following occurs:

|        |                                                                       |
|--------|-----------------------------------------------------------------------|
| EFAULT | <i>arg</i> points outside the allocated address space of the process. |
|--------|-----------------------------------------------------------------------|

**I\_FDINSERT**

creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

*arg* points to a *strfdinsert* structure which contains the following members:

|                             |                 |
|-----------------------------|-----------------|
| <b>struct</b> <i>strbuf</i> | <b>ctlbuf;</b>  |
| <b>struct</b> <i>strbuf</i> | <b>databuf;</b> |
| <b>long</b>                 | <b>flags;</b>   |
| <b>int</b>                  | <b>fd;</b>      |
| <b>int</b>                  | <b>offset;</b>  |

The *len* field in the *ctlbuf strbuf* structure (see *putmsg(2)*) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other stream and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I\_FDINSERT** will store a pointer to the *fd* stream's driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

*flags* specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to **RS\_HIPRI**. For non-priority messages, **I\_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I\_FDINSERT** does not block on this condition. For non-priority messages, **I\_FDINSERT** does not block when the write queue is full and **O\_NDELAY** is set. Instead, it fails and sets *errno* to **EAGAIN**.

**I\_FDINSERT** also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O\_NDELAY** has been specified. No partial message is sent.

**I\_FDINSERT** will fail if one of the following occurs:

|        |                                                                                                                                                                                                                           |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EAGAIN | A non-priority message was specified, the <b>O_NDELAY</b> flag is set, and the stream write queue is full due to internal flow control conditions.                                                                        |
| EAGAIN | Buffers could not be allocated for the message that was to be created.                                                                                                                                                    |
| EFAULT | <i>arg</i> points, or the buffer area specified in <i>ctlbuf</i> or <i>databuf</i> is, outside the allocated address space of the process.                                                                                |
| EINVAL | <i>fd</i> in the <i>strfdinsert</i> structure is not a valid, open stream file descriptor; the size of a pointer plus <i>offset</i> is greater than the <i>len</i> field for the buffer specified through <i>ctlptr</i> ; |



*offset* does not specify a properly-aligned location in the data buffer; an undefined value is pointed to by *flags*.

ENXIO

A hangup is received on the stream referred to by *fildev*.

ERANGE

The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I\_STR

Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to permit a process to specify timeouts and variable-sized amounts of data when sending an ioctl request to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. I\_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I\_STR can be active on a stream. Further I\_STR calls will block until the active I\_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O\_NDELAY (see *open(2V)*) flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioc* structure which contains the following members:

```
int    ic_cmd;        /* downstream command */
int    ic_timeout;    /* ACK/NAK timeout */
int    ic_len;        /* length of data arg */
char   *ic_dp;        /* ptr to data arg */
```

*ic\_cmd* is the internal ioctl command intended for a downstream module or driver and *ic\_timeout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I\_STR request will wait for acknowledgement before timing out. *ic\_len* is the number of bytes in the data argument and *ic\_dp* is a pointer to the data argument. The *ic\_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic\_dp* should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The *stream head* will convert the information pointed to by the *strioc* structure to an internal ioctl command message and send it downstream.

I\_STR will fail if one of the following occurs:

EAGAIN

Buffers could not be allocated for the ioctl message.

EFAULT

*arg* points, or the buffer area specified by *ic\_dp* and *ic\_len* (separately for data sent and data returned) is, outside the allocated address space of the process.

EINVAL

*ic\_len* is less than 0 or *ic\_len* is larger than the maximum

configured size of the data part of a message or *ic\_timeout* is less than -1.

- ENXIO                   A hangup is received on the stream referred to by *fildev*.
- ETIME                   A downstream *ioctl* timed out before acknowledgement was received.

An *I\_STR* can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *streamhead*. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the *ioctl* command sent downstream fails. For these cases, *I\_STR* will fail with *errno* set to the value in the message.

### **I\_SENDFD**

Requests the stream associated with *fildev* to send a message, containing a file pointer, to the *stream head* at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

*I\_SENDFD* converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see *intro(2)*) of the *stream head* at the other end of the stream pipe to which it is connected.

*I\_SENDFD* will fail if one of the following occurs:

- EAGAIN                   The sending stream is unable to allocate a message block to contain the file pointer.
- EAGAIN                   The read queue of the receiving *stream head* is full and cannot accept the message sent by *I\_SENDFD*.
- EBADF                   *arg* is not a valid, open file descriptor.
- EINVAL                   *fildev* is not connected to a stream pipe.
- ENXIO                   A hangup is received on the stream referred to by *fildev*.

### **I\_RECVFD**

Retrieves the file descriptor associated with the message sent by an *I\_SENDFD* *ioctl* over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

*fd* is an integer file descriptor. *uid* and *gid* are the user ID and group ID, respectively, of the sending stream.

If *O\_NDELAY* is not set (see *open(2V)*), *I\_RECVFD* will block until a message is present at the *streamhead*. If *O\_NDELAY* is set, *I\_RECVFD* will fail with *errno* set to *EAGAIN* if no message is present at the *streamhead*.

If the message at the *stream head* is a message sent by an *I\_SENDFD*, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*.

*I\_RECVFD* will fail if one of the following occurs:

- EAGAIN                   A message was not present at the *stream head* read queue, and the *O\_NDELAY* flag is set.



|         |                                                                                                         |
|---------|---------------------------------------------------------------------------------------------------------|
| EBADMSG | The message at the <i>stream head</i> read queue was not a message containing a passed file descriptor. |
| EFAULT  | <i>arg</i> points outside the allocated address space of the process.                                   |
| EMFILE  | Too many descriptors are active.                                                                        |
| ENXIO   | A hangup is received on the stream referred to by <i>fildev</i> .                                       |

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

#### I\_LINK

Connects two streams, where *fildev* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. I\_LINK causes the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see I\_UNLINK) on success, and a -1 on failure.

I\_LINK will fail if one of the following occurs:

|        |                                                                                                                                                                                               |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENXIO  | A hangup is received on the stream referred to by <i>fildev</i> .                                                                                                                             |
| ETIME  | The <i>ioctl</i> timed out before an acknowledgement was received.                                                                                                                            |
| EAGAIN | Storage could not be allocated to perform the I_LINK.                                                                                                                                         |
| EBADF  | <i>arg</i> is not a valid, open file descriptor.                                                                                                                                              |
| EINVAL | The stream referred to by <i>fildev</i> does not support multiplexing.                                                                                                                        |
| EINVAL | <i>arg</i> is not a stream, or is already linked under a multiplexor.                                                                                                                         |
| EINVAL | The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given <i>stream head</i> is linked into a multiplexing configuration in more than one place. |

An I\_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildev*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I\_LINK will fail with *errno* set to the value in the message.

#### I\_UNLINK

Disconnects the two streams specified by *fildev* and *arg*. *fildev* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the *ioctl* I\_LINK command when a stream was linked below the multiplexing driver. If *arg* is -1, then all streams which were linked to *fildev* are disconnected. As in I\_LINK, this command requires the multiplexing driver to acknowledge the unlink.

I\_UNLINK will fail if one of the following occurs:

|        |                                                                    |
|--------|--------------------------------------------------------------------|
| ENXIO  | A hangup is received on the stream referred to by <i>fildev</i> .  |
| ETIME  | The <i>ioctl</i> timed out before an acknowledgement was received. |
| EAGAIN | Buffers could not be allocated for the acknowledgement message.    |

**EINVAL**

The multiplexor ID number was invalid.

An **I\_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildev*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I\_UNLINK** will fail with *errno* set to the value in the message.

**SEE ALSO**

**close(2)**, **fcntl(2V)**, **getmsg(2)**, **intro(2)**, **ioctl(2)**, **open(2V)**, **poll(2)**, **putmsg(2)**, **read(2V)**, **sigvec(2)**, **write(2V)**

*STREAMS Programmer's Guide*

*STREAMS Primer*





**NAME**

intro – introduction to games and demos

**DESCRIPTION**

This section describes available games and demos.

**LIST OF GAMES AND DEMOS**

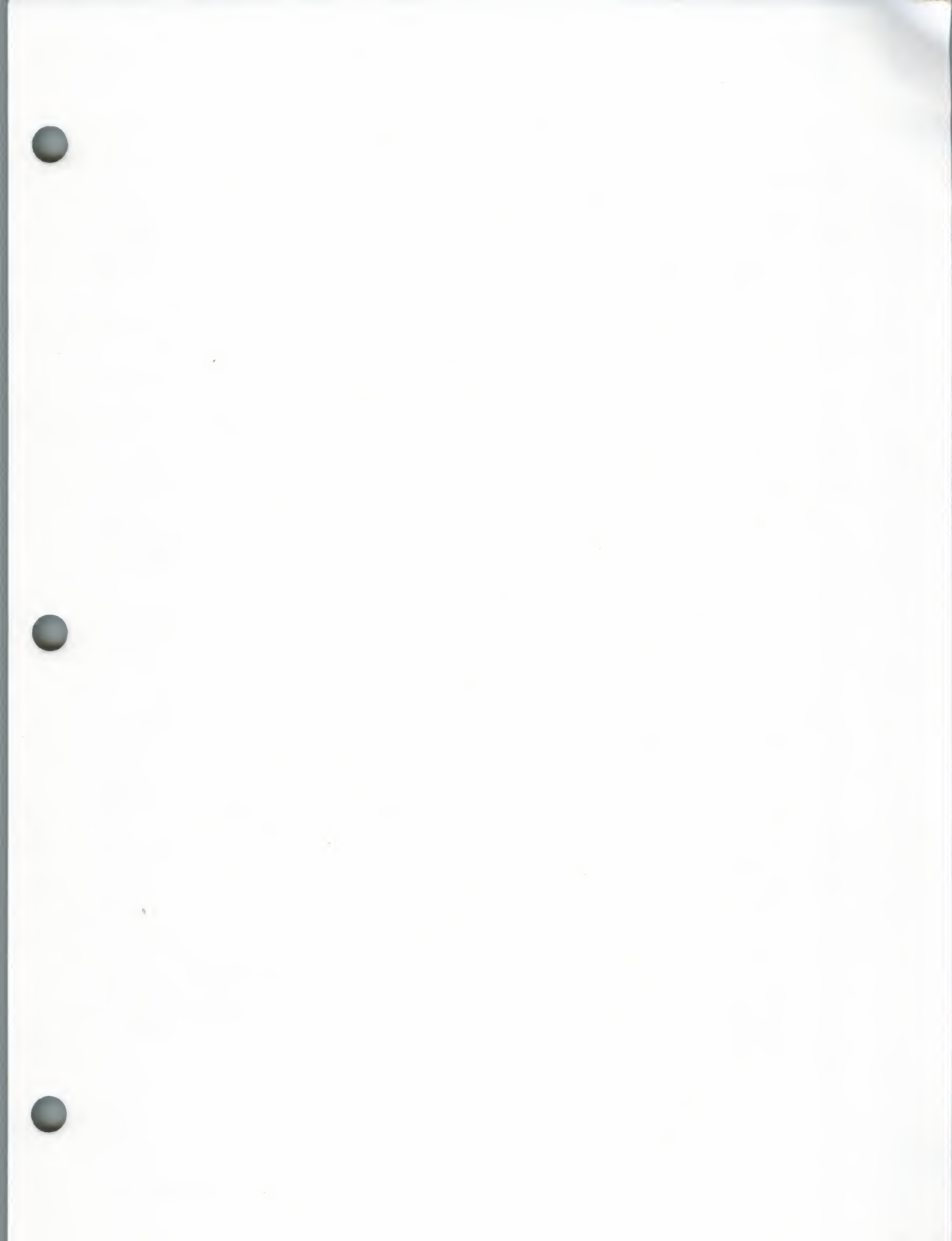
| Name           | Appears on Page   | Description                                      |
|----------------|-------------------|--------------------------------------------------|
| adventure      | adventure(6)      | an exploration game                              |
| arithmetic     | arithmetic(6)     | provide drill in number facts                    |
| backgammon     | backgammon(6)     | the game of backgammon                           |
| banner         | banner(6)         | print large banner on printer                    |
| battlestar     | battlestar(6)     | a tropical adventure game                        |
| bcd            | bcd(6)            | convert to antique media                         |
| bj             | bj(6)             | the game of black jack                           |
| boggle         | boggle(6)         | play the game of boggle                          |
| boggletool     | boggletool(6)     | play a game of boggle                            |
| bouncedemo     | graphics_demos(6) | graphics demonstration programs                  |
| canfield       | canfield(6)       | Canfield solitaire card game                     |
| canfieldtool   | canfield(6)       | Canfield solitaire card game                     |
| canvas_demo    | sunview_demos(6)  | Window-System demonstration programs             |
| cfscores       | canfield(6)       | Canfield solitaire card game                     |
| chase          | chase(6)          | try to escape to killer robots                   |
| chess          | chess(6)          | the game of chess                                |
| chesstool      | chesstool(6)      | window-based front-end to chess program          |
| ching          | ching(6)          | the book of changes and other cookies            |
| craps          | craps(6)          | the game of craps                                |
| cribbage       | cribbage(6)       | the card game cribbage                           |
| cursor_demo    | sunview_demos(6)  | Window-System demonstration programs             |
| dialtest       | dialtest(6)       | a demonstration and testing program for SunDials |
| factor         | factor(6)         | factor a number, generate large primes           |
| fish           | fish(6)           | play "Go Fish"                                   |
| fortune        | fortune(6)        | print a random, hopefully interesting, adage     |
| framedemo      | graphics_demos(6) | graphics demonstration programs                  |
| gammontool     | gammontool(6)     | play a game of backgammon                        |
| graphics_demos | graphics_demos(6) | graphics demonstration programs                  |
| hack           | hack(6)           | replacement for rogue                            |
| hangman        | hangman(6)        | computer version of the game hangman             |
| hunt           | hunt(6)           | a multiplayer multiterminal game                 |
| jumpdemo       | graphics_demos(6) | graphics demonstration programs                  |
| life           | life(6)           | John Conway's game of life                       |
| mille          | mille(6)          | play Mille Bornes                                |
| monop          | monop(6)          | Monopoly game                                    |
| moo            | moo(6)            | guessing game                                    |
| number         | number(6)         | convert Arabic numerals to English               |
| ppt            | bcd(6)            | convert to antique media                         |
| primes         | factor(6)         | factor a number, generate large primes           |
| primes         | primes(6)         | print all primes larger than some given number   |
| quiz           | quiz(6)           | test your knowledge                              |
| rain           | rain(6)           | animated raindrops display                       |
| random         | random(6)         | select lines randomly from a file                |
| robots         | robots(6)         | fight off villainous robots                      |
| snake          | snake(6)          | display chase game                               |
| snscore        | snake(6)          | display chase game                               |



spheresdemo  
sunview\_demos  
trek  
worm  
worms  
wump

graphics\_demos(6)  
sunview\_demos(6)  
trek(6)  
worm(6)  
worms(6)  
wump(6)

graphics demonstration programs  
Window-System demonstration programs  
trekkie game  
play the growing worm game  
animate worms on a display terminal  
the game of hunt-the-wumpus





## NAME

cribbage – the card game cribbage

## SYNOPSIS

/usr/games/cribbage [ -eqr ] name ...

## DESCRIPTION

**cribbage** plays the card game cribbage, with **cribbage** playing one hand and the user the other. **cribbage** initially asks the user if the rules of the game are needed – if so, **cribbage** displays the appropriate section from *According to Hoyle* with **more**(1).

## OPTIONS

- e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.
- q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.
- r Instead of asking the player to cut the deck, **cribbage** will randomly cut the deck.

## PLAYING CRIBBAGE

**cribbage** first asks the player whether he wishes to play a short game (“once around”, to 61) or a long game (“twice around”, to 121). A response of ‘s’ results in a short game, any other response plays a long game.

At the start of the first game, **cribbage** asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, **cribbage** first prints the player’s hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, **cribbage** cuts the deck (if it is the player’s crib) or asks the player to cut the deck (if it’s its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn’t have the crib) leading the first card. Play continues, as per **cribbage**, until all cards are exhausted. **cribbage** keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. **cribbage** requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

## SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *ranks* may be specified as one of a, 2, 3, 4, 5, 6, 7, 8, 9, t, j, q, and k, or alternatively, one of ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen, and king. *Suits* may be specified as s, h, d, and c, or alternatively as spades, hearts, diamonds, and clubs. A card may be specified as *rank suit*, or *rank of suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was 2h, 4d, 5c, 6h, jc, kd and you wanted to discard the king of diamonds, you could type any of k, king, kd, k d, k of d, king d, king of d, k diamonds, k of diamonds, king diamonds, or king of diamonds,

## FILES

/usr/games/cribbage

SEE ALSO  
more(1)





**NAME**

dialtest – a demonstration and testing program for SunDials

**SYNOPSIS**

/usr/demo/DIALS/dialtest

**DESCRIPTION**

dialtest displays a window with eight dials, corresponding to those on SunDials. To determine if the dialbox has been set up correctly, select the **Diagnostic** button on the panel. If the dialbox is correctly interfaced, **Dialbox OK** is displayed, and turning a dial on the box turn a dial on the screen. If **No Response** from Dialbox is displayed, repeat the dialbox install procedure.

**FILES**

/usr/demo/DIALS/dialtest



## NAME

factor, primes – factor a number, generate large primes

## SYNOPSIS

/usr/games/factor [ *number* ]

/usr/games/primes [ *number* ]

## DESCRIPTION

**factor** reads lines from its standard input. If it reads a positive number, **factor** will factor the number and print its prime factors, printing each one the proper number of times. **factor** exits when it reads zero, a negative number, or something other than a number. If a *number* is given, **factor** will factor the number, print its prime factors, and exit.

**primes** reads a number from the standard input and prints all primes larger than the given number and smaller than  $2^{32}$  (about  $4.3 \times 10^9$ ). If a *number* is given, **primes** will use that number rather than reading one from the standard input.

## DIAGNOSTICS

**Ouch.** Input out of range or for garbage input.

**NAME**

intro – introduction to system maintenance and operation commands

**DESCRIPTION**

This section contains information related to system bootstrapping, operation and maintenance. It describes all the server processes and daemons that run on the system, as well as standalone (PROM monitor) programs.

Disk formatting and labeling is done by **format(8S)**. Bootstrapping of the system is described in **boot(8S)** and **init(8)**. The standard set of commands run by the system when it boots is described in **rc(8)**. Related commands include those that check the consistency of file systems, **fsck(8)**; those that mount and unmount file systems, **mount(8)**; add swap devices, **swapon(8)**; force completion of outstanding file system I/O, **sync(2)**; shutdown or reboot a running system **shutdown(8)**, **halt(8)**, and **reboot(8)**; and, set the time on a machine from the time on another machine **rdate(8)**.

Creation of file systems is discussed in **mkfs(8)** and **newfs(8)**. File system performance parameters can be adjusted with **tunefs(8)**. File system backups and restores are described in **dump(8)** and **restore(8)**.

Procedures for adding new users to a system are described in **adduser(8)**, using **vipw(8)** to lock the password file during editing. **crash(8S)** which describes what happens when the system crashes, **savecore(8)** and **analyze(8)**, which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the **fsck(8)** file system repair program are **clri(8)**, **dcheck(8)**, **icheck(8)**, and **ncheck(8)**.

Configuring a new version of the kernel requires using the program **config(8)**; major system bootstraps often require the use of **mkproto(8)**. New devices are added to the **/dev** directory (once device drivers are configured into the system) using **makedev(8)** and **mknod(8)**. The **installboot(8S)** command can be used to install freshly compiled programs. The **catman(8)** command preformats the on-line manual pages.

Resource accounting is enabled by the **accton** command, and summarized by **sa(8)**. Login time accounting is performed by **ac(8)**. Disk quotas are managed using **quot(8)**, **quotacheck(8)**, **quotaon(8)**, and **repquota(8)**.

A number of servers and daemon processes are described in this section. The **update(8)** daemon forces delayed file system I/O to occur and **cron(8)** runs periodic events (such as removing temporary files from the disk periodically). The **syslogd(8)** daemon maintains the system error log. The **init(8)** process is the initial process created when the system boots. It manages the reboot process and creates the initial login prompts on the various system terminals, using **getty(8)**. The Internet super-server **inetd(8C)** invokes all other internet servers as needed. These servers include the remote shell servers **rshd(8C)** and **rexecd(8C)**, the remote login server **rlogind(8C)**, the FTP and TELNET daemons **ftpd(8C)**, and **telnetd(8C)**, the TFTP daemon **tftpd(8C)**, and the mail arrival notification daemon **comsat(8C)**. Other network daemons include the 'load average/who is logged in' daemon **rwhod(8C)**, the routing daemon **routed(8C)**, and the mail daemon **sendmail(8)**.

If network protocols are being debugged, then the protocol debugging trace program **trpt(8C)** is often useful. Remote magnetic tape access is provided by **rsh** and **rmt(8C)**. Remote line printer access is provided by **lpd(8)**, and control over the various print queues is provided by **lpc(8)**. Printer cost-accounting is done through **pac(8)**.

Network host tables may be gotten from the ARPA NIC using **gettable(8C)** and converted to UNIX-system-usable format using **htable(8)**.

**RPC and NFS daemons**

RPC and NFS daemons include:

|                |                                                                                    |
|----------------|------------------------------------------------------------------------------------|
| <b>portmap</b> | used by RPC based services.                                                        |
| <b>ypbind</b>  | used by the Yellow Pages to locate the Yellow Pages server.                        |
| <b>biod</b>    | used by NFS clients to read ahead to, and write behind from, network file systems. |
| <b>nfsd</b>    | the NFS server process that responds to NFS requests on NFS server machines.       |
| <b>ypserv</b>  | the Yellow Pages server, typically run on each NFS server.                         |
| <b>rstatd</b>  | the server counterpart of the remote speedometer tools.                            |



**mountd** the **mount** server that runs on NFS server machines and responds to requests by other machines to mount file systems.

**rwalld** used for broadcasting messages over the network.

## LIST OF MAINTENANCE COMMANDS

| Name                | Appears on Page        | Description                                                 |
|---------------------|------------------------|-------------------------------------------------------------|
| <b>ac</b>           | <b>ac(8)</b>           | login accounting                                            |
| <b>accton</b>       | <b>sa(8)</b>           | system accounting                                           |
| <b>adbgen</b>       | <b>adbgen(8)</b>       | generate adb script                                         |
| <b>adduser</b>      | <b>adduser(8)</b>      | procedure for adding new users                              |
| <b>arp</b>          | <b>arp(8C)</b>         | address resolution display and control                      |
| <b>audit</b>        | <b>audit(8)</b>        | audit trail maintenance                                     |
| <b>audit_warn</b>   | <b>audit_warn(8)</b>   | audit space low warning script                              |
| <b>auditd</b>       | <b>auditd(8)</b>       | audit daemon                                                |
| <b>automount</b>    | <b>automount(8)</b>    | automatically mount NFS file systems                        |
| <b>biod</b>         | <b>nfsd(8)</b>         | NFS daemons                                                 |
| <b>boot</b>         | <b>boot(8S)</b>        | start the system kernel or a standalone program             |
| <b>bootparamd</b>   | <b>bootparamd(8)</b>   | boot parameter server                                       |
| <b>captoinfo</b>    | <b>captoinfo(8V)</b>   | convert a termcap description into a terminfo description   |
| <b>catman</b>       | <b>catman(8)</b>       | create the cat files for the manual                         |
| <b>chown</b>        | <b>chown(8)</b>        | change owner                                                |
| <b>chroot</b>       | <b>chroot(8)</b>       | change root directory for a command                         |
| <b>client</b>       | <b>client(8)</b>       | add/remove diskless systems                                 |
| <b>cli</b>          | <b>cli(8)</b>          | clear i-node                                                |
| <b>comsat</b>       | <b>comsat(8C)</b>      | biff server                                                 |
| <b>config</b>       | <b>config(8)</b>       | build system configuration files                            |
| <b>crash</b>        | <b>crash(8S)</b>       | what happens when the system crashes                        |
| <b>cron</b>         | <b>cron(8)</b>         | clock daemon                                                |
| <b>dcheck</b>       | <b>dcheck(8)</b>       | file system directory consistency check                     |
| <b>dinfo</b>        | <b>dinfo(8)</b>        | report information about a disk's geometry and partitioning |
| <b>dmesg</b>        | <b>dmesg(8)</b>        | collect system diagnostic messages to form error log        |
| <b>dump</b>         | <b>dump(8)</b>         | incremental file system dump                                |
| <b>dumpfs</b>       | <b>dumpfs(8)</b>       | dump file system information                                |
| <b>edquota</b>      | <b>edquota(8)</b>      | edit user quotas                                            |
| <b>eeprom</b>       | <b>eeprom(8S)</b>      | Sun-3 EEPROM display and load utility                       |
| <b>etherd</b>       | <b>etherd(8C)</b>      | Ethernet statistics server                                  |
| <b>etherfind</b>    | <b>etherfind(8C)</b>   | find packets on Ethernet                                    |
| <b>exportfs</b>     | <b>exportfs(8)</b>     | export and unexport directories to NFS clients              |
| <b>fastboot</b>     | <b>fastboot(8)</b>     | reboot/halt the system without checking the disks           |
| <b>fasthalt</b>     | <b>fastboot(8)</b>     | reboot/halt the system without checking the disks           |
| <b>fingerd</b>      | <b>fingerd(8C)</b>     | remote user information server                              |
| <b>format</b>       | <b>format(8S)</b>      | disk partitioning and maintenance utility                   |
| <b>fpa_download</b> | <b>fpa_download(8)</b> | download to the Floating Point Accelerator (FPA)            |
| <b>fparel</b>       | <b>fparel(8)</b>       | Sun FPA online reliability tests                            |
| <b>fpaversion</b>   | <b>fpaversion(8)</b>   | print FPA version                                           |
| <b>fpurel</b>       | <b>fpurel(8)</b>       | perform tests the Sun Floating Point Co-processor           |
| <b>fpuversion4</b>  | <b>fpuversion4(8)</b>  | print the Sun-4 FPU version                                 |
| <b>fsck</b>         | <b>fsck(8)</b>         | file system consistency check and interactive repair        |
| <b>fsirand</b>      | <b>fsirand(8)</b>      | install random inode generation numbers                     |
| <b>ftpd</b>         | <b>ftpd(8C)</b>        | DARPA Internet File Transfer Protocol server                |
| <b>gettable</b>     | <b>gettable(8C)</b>    | get DoD Internet format host table from a host              |
| <b>getty</b>        | <b>getty(8)</b>        | set terminal mode                                           |
| <b>gpconfig</b>     | <b>gpconfig(8)</b>     | initialize the Graphics Processor                           |

|                       |                          |                                                           |
|-----------------------|--------------------------|-----------------------------------------------------------|
| <b>grpck</b>          | <b>grpck(8)</b>          | check group database entries                              |
| <b>halt</b>           | <b>halt(8)</b>           | stop the processor                                        |
| <b>htable</b>         | <b>htable(8)</b>         | convert DoD Internet format host table                    |
| <b>icheck</b>         | <b>icheck(8)</b>         | file system storage consistency check                     |
| <b>ifconfig</b>       | <b>ifconfig(8C)</b>      | configure network interface parameters                    |
| <b>inetd</b>          | <b>inetd(8C)</b>         | Internet services daemon                                  |
| <b>infocmp</b>        | <b>infocmp(8V)</b>       | compare or print out terminfo descriptions                |
| <b>init</b>           | <b>init(8)</b>           | process control initialization                            |
| <b>installboot</b>    | <b>boot(8S)</b>          | start the system kernel or a standalone program           |
| <b>iostat</b>         | <b>iostat(8)</b>         | report I/O statistics                                     |
| <b>ipallocald</b>     | <b>ipallocald(8C)</b>    | Ethernet-to-IP address allocator                          |
| <b>kadb</b>           | <b>kadb(8S)</b>          | adb-like kernel and standalone-program debugger           |
| <b>keyenvoy</b>       | <b>keyenvoy(8C)</b>      | talk to keyserver                                         |
| <b>keyserv</b>        | <b>keyserv(8C)</b>       | server for storing public and private keys                |
| <b>kgmon</b>          | <b>kgmon(8)</b>          | generate a dump of the operating system's profile buffers |
| <b>ldconfig</b>       | <b>ldconfig(8)</b>       | configure cache for <b>ld.so</b>                          |
| <b>link</b>           | <b>link(8)</b>           | exercise link and unlink system calls                     |
| <b>lockd</b>          | <b>lockd(8C)</b>         | network lock daemon                                       |
| <b>logintool</b>      | <b>logintool(8)</b>      | graphic login interface                                   |
| <b>lpc</b>            | <b>lpc(8)</b>            | line printer control program                              |
| <b>lpd</b>            | <b>lpd(8)</b>            | printer daemon                                            |
| <b>mailstats</b>      | <b>mailstats(8)</b>      | print statistics collected by sendmail                    |
| <b>MAKEDBM</b>        | <b>makedbm(8)</b>        | make a Yellow Pages dbm file                              |
| <b>MAKEDEV</b>        | <b>makedev(8)</b>        | make system special files                                 |
| <b>makekey</b>        | <b>makekey(8)</b>        | generate encryption key                                   |
| <b>mc68881version</b> | <b>mc68881version(8)</b> | print the MC68881 mask number and approximate clock rate  |
| <b>mconnect</b>       | <b>mconnect(8)</b>       | connect to SMTP mail server socket                        |
| <b>mkfs</b>           | <b>mkfs(8)</b>           | construct a file system                                   |
| <b>mknod</b>          | <b>mknod(8)</b>          | build special file                                        |
| <b>mkproto</b>        | <b>mkproto(8)</b>        | construct a prototype file system                         |
| <b>modload</b>        | <b>modload(8)</b>        | load a loadable module                                    |
| <b>modstat</b>        | <b>modstat(8)</b>        | display status of loadable modules                        |
| <b>modunload</b>      | <b>modunload(8)</b>      | unload a loadable module                                  |
| <b>monitor</b>        | <b>monitor(8S)</b>       | system ROM monitor                                        |
| <b>mount</b>          | <b>mount(8)</b>          | mount and dismount filesystems                            |
| <b>mountd</b>         | <b>mountd(8C)</b>        | NFS mount request server                                  |
| <b>named</b>          | <b>named(8C)</b>         | Internet domain name server                               |
| <b>ncheck</b>         | <b>ncheck(8)</b>         | generate names from i-numbers                             |
| <b>ndbootd</b>        | <b>ndbootd(8C)</b>       | ND boot block server                                      |
| <b>netconfig</b>      | <b>netconfig(8C)</b>     | PNP boot service                                          |
| <b>netstat</b>        | <b>netstat(8C)</b>       | show network status                                       |
| <b>newaliases</b>     | <b>newaliases(8)</b>     | rebuild the data base for the mail aliases file           |
| <b>newfs</b>          | <b>newfs(8)</b>          | construct a new file system                               |
| <b>newkey</b>         | <b>newkey(8)</b>         | create a new key in the publickey database                |
| <b>nfsd</b>           | <b>nfsd(8)</b>           | NFS daemons                                               |
| <b>nfsstat</b>        | <b>nfsstat(8C)</b>       | Network File System statistics                            |
| <b>pac</b>            | <b>pac(8)</b>            | printer/plotter accounting information                    |
| <b>ping</b>           | <b>ping(8C)</b>          | send ICMP ECHO_REQUEST packets to network hosts           |
| <b>pnps386</b>        | <b>pnps386(8C)</b>       | PNP diskless boot service                                 |
| <b>pnpsboot</b>       | <b>pnpsboot(8C)</b>      | PNP diskless boot service                                 |
| <b>pnpd</b>           | <b>pnpd(8C)</b>          | PNP daemon                                                |
| <b>portmap</b>        | <b>portmap(8C)</b>       | DARPA port to RPC program number mapper                   |
| <b>praudit</b>        | <b>praudit(8)</b>        | print contents of an audit trail file                     |



|              |                 |                                                   |
|--------------|-----------------|---------------------------------------------------|
| pstat        | pstat(8)        | print system facts                                |
| pwck         | pwck(8)         | check password database entries                   |
| pwdauthd     | pwdauthd(8C)    | server for authenticating passwords               |
| quot         | quot(8)         | summarize file system ownership                   |
| quotacheck   | quotacheck(8)   | file system quota consistency checker             |
| quotaoff     | quotaon(8)      | turn file system quotas on and off                |
| quotaon      | quotaon(8)      | turn file system quotas on and off                |
| rarpd        | rarpd(8C)       | DARPA Reverse Address Resolution Protocol service |
| rc.boot      | rc(8)           | command scripts for auto-reboot and daemons       |
| rc.local     | rc(8)           | command scripts for auto-reboot and daemons       |
| rc           | rc(8)           | command scripts for auto-reboot and daemons       |
| rdate        | rdate(8C)       | set system date from a remote host                |
| rdump        | dump(8)         | incremental file system dump                      |
| reboot       | reboot(8)       | restart the operating system                      |
| renice       | renice(8)       | alter priority of running processes               |
| repquota     | repquota(8)     | summarize quotas for a file system                |
| restore      | restore(8)      | incremental file system restore                   |
| rex          | rex(8C)         | RPC-based remote execution server                 |
| rexecd       | rexecd(8C)      | remote execution server                           |
| rlogind      | rlogind(8C)     | remote login server                               |
| rmail        | rmail(8C)       | handle remote mail received via uucp              |
| rmt          | rmt(8C)         | remote magtape protocol module                    |
| route        | route(8C)       | manually manipulate the routing tables            |
| routed       | routed(8C)      | network routing daemon                            |
| rpcinfo      | rpcinfo(8C)     | report RPC information                            |
| rquotad      | rquotad(8C)     | remote quota server                               |
| rrestore     | restore(8)      | incremental file system restore                   |
| rshd         | rshd(8C)        | remote shell server                               |
| rstatd       | rstatd(8C)      | kernel statistics server                          |
| usersd       | usersd(8C)      | network username server                           |
| rwalld       | rwalld(8C)      | network rwall server                              |
| rwhod        | rwhod(8C)       | system status server                              |
| sa           | sa(8)           | system accounting                                 |
| savecore     | savecore(8)     | save a core dump of the operating system          |
| sendmail     | sendmail(8)     | send mail over the internet                       |
| setup_client | setup_client(8) | create or remove a nfs client on a 4.0 server.    |
| setup_exec   | setup_exec(8)   | install architecture-dependent executable files   |
| showmount    | showmount(8)    | show all remote mounts                            |
| shutdown     | shutdown(8)     | close down the system at a given time             |
| spray        | spray(8C)       | spray packets                                     |
| sprayd       | sprayd(8C)      | spray server                                      |
| statd        | statd(8C)       | network status monitor                            |
| sticky       | sticky(8)       | persistent text and append-only directories       |
| sunddiag     | sunddiag(8)     | system diagnostics                                |
| suninstall   | suninstall(8)   | SunOS software installation program               |
| sunupgrade   | sunupgrade(8)   | upgrade the Sun Operating System                  |
| swapon       | swapon(8)       | specify additional device for paging and swapping |
| sysdiag      | sysdiag(8)      | system diagnostics                                |
| syslogd      | syslogd(8)      | log system messages                               |
| talkd        | talkd(8C)       | server for talk program                           |
| telnetd      | telnetd(8C)     | DARPA TELNET protocol server                      |
| tftpd        | tftpd(8C)       | DARPA Trivial File Transfer Protocol server       |
| tic          | tic(8V)         | terminfo compiler                                 |

timed  
 tnamed  
 trpt  
 tunefts  
 umount  
 unconfigure  
 unlink  
 update  
 uuclean  
 vipw  
 vmstat  
 ypbind  
 ypinit  
 ypmake  
 yppasswdd  
 yppoll  
 yppush  
 ypserv  
 ypset  
 ypupdated  
 ypwhich  
 ypxfr  
 zdump  
 zic

timed(8C)  
 tnamed(8C)  
 trpt(8C)  
 tunefts(8)  
 mount(8)  
 unconfigure(8)  
 link(8)  
 update(8)  
 uuclean(8C)  
 vipw(8)  
 vmstat(8)  
 ypserv(8)  
 ypinit(8)  
 ypmake(8)  
 yppasswdd(8C)  
 yppoll(8)  
 yppush(8)  
 ypserv(8)  
 ypset(8)  
 ypupdated(8C)  
 ypwhich(8)  
 ypxfr(8)  
 zdump(8)  
 zic(8)

DARPA Time server  
 DARPA Trivial name server  
 transliterate protocol trace  
 tune up an existing file system  
 mount and dismount filesystems  
 reset the network configuration for a system  
 exercise link and unlink system calls  
 periodically update the super block  
 uucp spool directory clean-up  
 edit the password file  
 report virtual memory statistics  
 Yellow Pages server and binder processes  
 build and install Yellow Pages database  
 rebuild Yellow Pages database  
 server for modifying Yellow Pages password file  
 what version of a YP map is at a YP server host  
 force propagation of a changed YP map  
 Yellow Pages server and binder processes  
 point ypbind at a particular server  
 server for changing yp information  
 what machine is the YP server?  
 transfer YP map from a YP server to here  
 time zone dumper  
 time zone compiler



**NAME**

ac - login accounting

**SYNOPSIS**

/usr/etc/ac [ -w *wtmp* ] [ -p ] [ -d ] [ *username* ] ...

**DESCRIPTION**

ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file /var/adm/wtmp is maintained by *init*(8) and *login*(1). Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

**OPTIONS**

- w *wtmp* Specify an alternate *wtmp* file.
- p Print individual totals; without this option, only totals are printed.
- d Printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, /var/adm/wtmp is used.

**FILES**

/var/adm/wtmp

**SEE ALSO**

*login*(1), *utmp*(5), *init*(8), *sa*(8)





**NAME**

cron – clock daemon

**SYNOPSIS**

/usr/etc/cron

**DESCRIPTION**

**cron** executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in **crontab** files in the directory **/var/spool/cron/crontabs**. Users can submit their own **crontab** files using the **crontab(1)** command. Commands that are to be executed only once may be submitted using the **at(1)** command.

**cron** only examines **crontab** files and **at** command files during process initialization and when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since **cron** never exits, it should only be executed once. This is normally done by running **cron** from the initialization process through the file **/etc/rc**; see **init(8)**. **/var/spool/cron/FIFO** is a FIFO file that **crontab** and **at** use to communicate with **cron**; it is also used as a lock file to prevent the execution of more than one **cron**.

**FILES**

|                                 |                                           |
|---------------------------------|-------------------------------------------|
| <b>/var/spool/cron</b>          | main cron directory                       |
| <b>/var/spool/cron/FIFO</b>     | FIFO for sending messages to <b>cron</b>  |
| <b>/var/spool/cron/crontabs</b> | directory containing <b>crontab</b> files |

**SEE ALSO**

**at(1)**, **crontab(1)**, **sh(1)**, **queuedefs(5)**, **init(8)**, **syslogd(8)**

**DIAGNOSTICS**

**cron** logs various errors to the system log daemon, **syslogd(8)**, with a facility code of **cron**. The messages are listed here, grouped by severity level.

**Err Severity**

**Can't create /var/spool/cron/FIFO: reason**

**cron** was unable to start up because it could not create **/var/spool/cron/FIFO**.

**Can't access /var/spool/cron/FIFO: reason**

**cron** was unable to start up because it could not access **/var/spool/cron/FIFO**.

**Can't open /var/spool/cron/FIFO: reason**

**cron** was unable to start up because it could not open **/var/spool/cron/FIFO**.

**Can't start cron - another cron may be running (/var/spool/cron/FIFO exists)**

**cron** found that **/var/spool/cron/FIFO** already existed when it was started; this normally means that **cron** had already been started, but it may mean that an earlier **cron** terminated abnormally without removing **/var/spool/cron/FIFO**.

**Can't stat /var/spool/cron/FIFO: reason**

**cron** could not get the status of **/var/spool/cron/FIFO**.

**Can't change directory to directory:reason**

**cron** could not change to **directory**.

**Can't read directory:reason**

**cron** could not read **directory**.

**error reading message: reason**

An error occurred when **cron** tried to read a control message from **/var/spool/cron/FIFO**.

**message received — bad format**

A message was successfully read by **cron** from **/var/spool/cron/FIFO**, but the message was not of a form recognized by **cron**.

**SIGTERM**

received **cron** was told to terminate by having a SIGTERM signal sent to it.

**cron could not unlink /var/spool/cron/FIFO: reason**

**cron** was told to terminate, but it was unable to unlink **/var/spool/cron/FIFO** before it terminated.

**\*\*\*\*\* CRON ABORTED \*\*\*\*\***

**cron** terminated, either due to an error or because it was told to.

**Can't open queuedefs file file:reason**

**cron** could not open a *queuedefs* file.

**I/O error reading queuedefs file file:reason**

An I/O error occurred while **cron** was reading a *queuedefs* file.

**Using default queue definitions**

An error occurred while trying to read a *queuedefs* file; the default queue definitions will be used.

**Can't allocate numberbytes of space**

An internal error occurred in **cron** while trying to allocate memory.

**Info Severity****queue queue max run limit reached**

There were more jobs running or to be run in the queue *queue* than the maximum number specified. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

**MAXRUN (25) procs reached**

There were more than 25 jobs running or to be run by **cron**. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

**\*\*\* cron started \*\*\***

**cron** started running.

**> CMD: command**

A **cron** job was started. *command* is the command to be run.

**> user pid queue time**

A **cron** job was started for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*.

**< user pid queue time status**

A **cron** job completed for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. If the command terminated with a non-zero exit status or a signal, *status* indicates the exit status or signal.

**Notice Severity****Can't fork**

An attempt to fork (2) to run a new job failed; **cron** will attempt again after a 30-second delay.

**Warning Severity****Can't stat queuedefs file file:reason**

**cron** could not get the status of a *queuedefs* file in order to determine whether it has changed. **cron** will assume it has changed and will reread it.





**NAME**

**dbconfig** – initializes the dial box

**SYOPNSIS**

**/usr/etc/dbconfig**

**DESCRIPTION**

**dbconfig** opens the designated serial port and sets its baud, parity and transmission rates. It also removes all STREAMS modules already pushed upon it (such as **ttcompat(4M)** and **ldterm(4M)**) and pushes the dial box STREAMS module “**db**” onto the device. **db** then holds the stream open to maintain this configuration. If the device **/dev/dialbox** has not been created and linked to the serial port, **dbconfig** will fail.

**FILES**

**/dev/dialbox**

**SEE ALSO**

**db(4M)**, **ldterm(4M)**, **ttcompat(4M)**



**NAME**

**dcheck** – file system directory consistency check

**SYNOPSIS**

**/usr/etc/dcheck** [ *-i numbers* ] [ *filesystem* ]

**DESCRIPTION**

**Note:** **dcheck** has been superceded for normal consistency checking by **fsck(8)**.

**dcheck** reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, **dcheck** checks a set of default file systems.

**dcheck** is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

**OPTIONS**

**-i numbers**

*numbers* is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

**FILES**

Default file systems vary with installation.

**SEE ALSO**

**fs(5)**, **fsck(8)**, **clri(8)**, **icheck(8)**, **ncheck(8)**

**DIAGNOSTICS**

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

**BUGS**

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

**NAME**

**fpa\_download** – download to the Floating Point Accelerator (FPA)

**SYNOPSIS**

**fpa\_download** [ **-d** ] [ **-r** ] [ **-v** ] [ **-u** *ufile* ] [ **-m** *mfile* ] [ **-c** *cfile* ]

**AVAILABILITY**

**fpa\_download**(8) applies to Sun3 and Sun3x systems equipped with either an FPA or FPA+.

**DESCRIPTION**

**fpa\_download** writes microcode, map, and constants files to FPA and FPA+ boards. FPA requires a map file; FPA+ does not.

Root execution level is required to download (d,u,m and c options). **fpa\_download** is called from */etc/rc.local* when */dev/fpa* exists.

Given no arguments, **fpa\_download** prints whether an FPA, or FPA+ is installed.

**OPTIONS**

- d** Download microcode, constants, and map files. Enable default file names.
- r** Print microcode and constant revision.
- v** Verbose mode.
- u** *ufile* Download microcode from *ufile*.
- m** *mfile* Download map from *mfile* (FPA only).
- c** *cfile* Download constants from *cfile*.

**FILES**

|                                    |                                         |
|------------------------------------|-----------------------------------------|
| <i>/dev/fpa</i>                    | device file for both FPA and FPA+.      |
| <i>/usr/etc/fpa/fpa_micro_bin</i>  | default microcode file (ufile) for FPA. |
| <i>/usr/etc/fpa/fpa_constants</i>  | default constants file (cfile) for FPA  |
| <i>/usr/etc/fpa/fpa_micro_map</i>  | default map file (mfile) for FPA        |
| <i>/usr/etc/fpa/fpa_micro_bin+</i> | default microcode file (ufile) for FPA+ |
| <i>/usr/etc/fpa/fpa_constants+</i> | default constants file (cfile) for FPA+ |

**SEE ALSO**

**fpa**(4)

**DIAGNOSTICS**

The following diagnostics are printed when **fpa\_download** encounters a serious error and asks the kernel to disable the FPA. This might occur if the microcode, map, or constants files are corrupted, or if there is an FPA or system hardware problem.

**FPA Download Failed - FPA ioctl failed**

An ioctl() on */dev/fpa* failed, possibly due to a hung FPA pipe.

**FPA Failed Download - FPA Bus Error**

Received a SIGFPE.

**FPA Failed Download - Upload mismatch**

After each file is written to the FPA/FPA+, **fpa\_download** uploads the contents of FPA memory and compares it with the source. They should always match.





**NAME**

**fparel** - Sun FPA online reliability tests

**SYNOPSIS**

**fparel** [ **-pn** ] [ **-v** ]

**AVAILABILITY**

Not available on Sun386i systems.

**DESCRIPTION**

**fparel** is a command to execute the Sun FPA online confidence and reliability test program. **fparel** tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. **fparel** runs without disturbing other processes that may be using the FPA. **fparel** can only be run by the super-user.

After a successful pass, **fparel** writes

**time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31**

to the file **/var/adm/diaglog**.

If a pass fails, **fparel** writes

**time, date: Sun FPA failed**

along with the test name and context number that failed, to the file **/var/adm/diaglog**. **fparel** then broadcasts the message

**time, date: Sun FPA failed, disabled, service required**

to all users of the system. Next, **fparel** causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file **/etc/rc.local** should contain an entry to cause **fparel** to be invoked upon reboot to be sure that the FPA remains inaccessible in cases where rebooting doesn't correct the problem. See **rc(8)**.

The **crontab(5)** file for root should contain an entry indicating that **cron(8)** is to run **fparel** daily, such as:

**7 2 \* \* \* /usr/etc/fpa/fparel**

which causes **fparel** to run at seven minutes past two, every day. See **cron(8)** and **crontab(5)** for details.

**OPTIONS**

**-pn** Perform *n* passes. Default is *n*=1. **-p0** means perform 2147483647 passes.

**-v** Run in verbose mode with detailed test results to the standard output.

**FILES**

**/var/adm/diaglog** Log of **fparel** diagnostics.

**/etc/rc.local**

**/var/spool/cron/crontabs/root**

**/usr/etc/fpa/\*** directory containing FPA microcode, data files, and loader

**SEE ALSO**

**fpaversion(8)**, **crontab(5)**, **cron(8)**, **rc(8)**



**NAME**

**fpaversion** – print the Floating Point Accelerator (FPA) version

**SYNOPSIS**

**fpaversion** [ **-hlqv** ] [ **-t** [ **cdhimprstvx**CIMS ] ]

**AVAILABILITY**

**fpaversion(8)** applies to Sun3 and Sun3x systems equipped with either an FPA or FPA+.

**DESCRIPTION**

**fpaversion** performs various tests on the FPA or FPA+. When given no arguments, it prints the microcode version number and constants currently installed on **/dev/fpa**. **fpaversion** also performs a quick test to ensure proper operation and reports whether an FPA or an FPA+ is installed.

**OPTIONS**

- h** Help. Print command-line summary.
- l** Loop through tests infinitely.
- q** Quiet output. Print out only error messages.
- v** Verbose output.
- t** Specify certain tests:
  - c** Command register format instructions.
  - d** Double precision format instructions.
  - h** Help. Print summary of test specifiers.
  - i** Imask register.
  - m** Mode register.
  - p** Simple pipe sequencing.
  - r** User registers for all contexts.
  - s** Single precision format instructions.
  - t** Status generation.
  - v** Print version number and date of microcode, and constants. Report whether an FPA or FPA+ is installed.
  - x** Extended format instructions.
  - C** Check checksum for microcode, mapping RAM, and constant RAM for the FPA. Check checksum for microcode RAM and constant RAM for the FPA+.
  - I** Allows interactive reads and writes to the FPA.
  - M** Command register format matrix instructions.
  - S** Shadow registers.

**FILES**

|                                    |                                    |
|------------------------------------|------------------------------------|
| <b>/dev/fpa</b>                    | physical FPA device                |
| <b>/usr/etc/fpa/fpa_micro_bin</b>  | microcode binaries for the FPA     |
| <b>/usr/etc/fpa/fpa_micro_map</b>  | microcode map binaries for the FPA |
| <b>/usr/etc/fpa/fpa_constants</b>  | microcode data file for the FPA    |
| <b>/usr/etc/fpa/fpa_micro_bin+</b> | microcode binaries for the FPA+    |
| <b>/usr/etc/fpa/fpa_constants+</b> | microcode data file for the FPA+   |
| <b>/usr/etc/fpa/fpa_download</b>   | microcode loader                   |

**SEE ALSO****fparel(8), sysdiag(8)****DIAGNOSTICS**

If a test fails, its name, along with the actual and expected results will be printed.



**NAME**

**fpurel** – perform tests the Sun Floating Point Co-processor

**SYNOPSIS**

**fpurel** [ **-rv** ] [ **-p**[*count*] ]

**DESCRIPTION**

**fpurel** performs a series of functional and computational tests for the Sun Floating Point Co-processor to verify that it is operational and accurate. With no options, **fpurel** runs one pass silently in the foreground and only reports errors if any are found.

**OPTIONS**

- r** Disable stop on error. Continue to run if errors are detected. The default is to display the error message and to stop testing when an error is detected.
- v** Verbose. Display the name and results of each test on the console. The default is to run silently.
- p** [*count*] Pass count. Specify the number of times to run the test suite. The default is to run one pass.

**EXAMPLE**

This example uses **fpurel** from the **/usr/diag** directory. If no errors are detected, then no information is displayed.

**% /usr/diag/fpurel**

**NAME**

**fpuversion4** – print the Sun-4 FPU version

**SYNOPSIS**

**/usr/etc/fpuversion4**

**AVAILABILITY**

Sun-4 systems only.

**DESCRIPTION**

**fpuversion4** reads the `%fsr` register to determine the FPU version installed on a Sun-4. The printed version field contains a value in the range 0-7; by SPARC convention 7 indicates that no FPU is installed, so floating-point instructions are always emulated in the kernel.





## NAME

mount, umount – mount and unmount filesystems

## SYNOPSIS

```
/usr/etc/mount [ -p ]
/usr/etc/mount -a[fnv] [ -t type ]
/usr/etc/mount [ -fnrv ] [ -t type ] [ -o options ] filesystem directory
/usr/etc/mount [ -vfn ] [ -o options ] filesystem | directory

/usr/etc/umount [ -t type ] [ -h host ]
/usr/etc/umount -a[v]
/usr/etc/umount [ -v ] filesystem | directory ...
```

## DESCRIPTION

**mount** attaches a named *filesystem* to the filesystem hierarchy at the pathname location *directory*, which must already exist. If *directory* has any contents prior to the **mount** operation, these remain hidden until the *filesystem* is once again unmounted. If *filesystem* is of the form *host:pathname*, it is assumed to be an NFS filesystem (type *nfs*).

**umount** unmounts a currently mounted filesystem, which can be specified either as a *directory* or a *filesystem*.

**mount** and **umount** maintain a table of mounted filesystems in */etc/mtab*, described in *fstab(5)*. If invoked without an argument, **mount** displays the contents of this table. If invoked with either a *filesystem* or *directory* only, **mount** searches the file */etc/fstab* for a matching entry, and mounts the filesystem indicated in that entry on the indicated directory.

## MOUNT OPTIONS

- p** Print the list of mounted filesystems in a format suitable for use in */etc/fstab*.
- a** All. Attempt to mount all the filesystems described in */etc/fstab*. If a *type* argument is specified with **-t**, mount all filesystems of that type. Using **-a**, **mount** builds a dependency tree of mount points in */etc/fstab*. **mount** will correctly mount these filesystems regardless of their order in */etc/fstab* (except loopback mounts; see WARNINGS below).
- f** Fake an */etc/mtab* entry, but do not actually mount any filesystems.
- n** Mount the filesystem without making an entry in */etc/mtab*.
- v** Verbose. Display a message indicating each filesystem being mounted.
- t type** Specify a filesystem type. The accepted types are *4.2*, *nfs*, and *lo*. See *fstab(5)* for a description of *4.2*, and *nfs*; see *lofs(4S)* for a description of *lo*.
- r** Mount the specified filesystem read-only, even if the entry in */etc/fstab* specifies that it is to be mounted read-write.

Physically write-protected and magnetic-tape filesystems must be mounted read-only. Otherwise errors occur when the system attempts to update access times, even if no write operation is attempted.

**-o options**

Specify filesystem *options* —list of comma-separated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

*options* valid on *all* filesystems:

|                    |                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>rw ro</b>       | Read/write or read-only.                                                                                                                                                                             |
| <b>suid nosuid</b> | Setuid execution allowed or disallowed.                                                                                                                                                              |
| <b>grpuid</b>      | Create files with BSD semantics for the propagation of the group ID. Under this option, files inherit the GID of the directory in which they are created, regardless of the directory's set-GID bit. |



|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>noauto</b>  | Do not mount this filesystem that is currently mounted read-only. If the filesystem is not currently mounted, an error results.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>remount</b> | If the file system is currently mounted, and if the entry in <i>/etc/fstab</i> specifies that it is to be mounted read-write or <b>rw</b> was specified along with <b>remount</b> , remount the file system making it read-write. If the entry in <i>/etc/fstab</i> specifies that it is to be mounted read-only and <b>rw</b> was not specified, the file system is not remounted. If the file system is currently mounted read-write, specifying <b>ro</b> along with <b>remount</b> results in an error. If the file system is not currently mounted, an error results. |

The default is '**rw,suid**'.

*options* specific to 4.2 filesystems:

**quota|noquota** Usage limits are enforced, or are not enforced. The default is **noquota**.

*options* specific to **nfs** (NFS) filesystems:

|                          |                                                                                                          |
|--------------------------|----------------------------------------------------------------------------------------------------------|
| <b>bg fg</b>             | If the first attempt fails, retry in the background, or, in the foreground.                              |
| <b>retry=<i>n</i></b>    | The number of times to retry the mount operation.                                                        |
| <b>rsize=<i>n</i></b>    | Set the read buffer size to <i>n</i> bytes.                                                              |
| <b>wsiz=<i>n</i></b>     | Set the write buffer size to <i>n</i> bytes.                                                             |
| <b>timeo=<i>n</i></b>    | Set the NFS timeout to <i>n</i> tenths of a second.                                                      |
| <b>retrans=<i>n</i></b>  | The number of NFS retransmissions.                                                                       |
| <b>port=<i>n</i></b>     | The server IP port number.                                                                               |
| <b>soft hard</b>         | Return an error if the server does not respond, or continue the retry request until the server responds. |
| <b>intr</b>              | Allow keyboard interrupts on hard mounts.                                                                |
| <b>secure</b>            | Use a more secure protocol for NFS transactions.                                                         |
| <b>acregmin=<i>n</i></b> | Hold cached attributes for at least <i>n</i> seconds after file modification.                            |
| <b>acregmax=<i>n</i></b> | Hold cached attributes for no more than <i>n</i> seconds after file modification.                        |
| <b>acdirmin=<i>n</i></b> | Hold cached attributes for at least <i>n</i> seconds after directory update.                             |
| <b>acdirmax=<i>n</i></b> | Hold cached attributes for no more than <i>n</i> seconds after directory update.                         |
| <b>actimeo=<i>n</i></b>  | Set <i>min</i> and <i>max</i> times for regular files and directories to <i>n</i> seconds.               |
| <b>noac</b>              | Suppress attribute caching.                                                                              |

Regular defaults are:

**fg, retry=10000, timeo=7, retrans=3, port=NFS\_PORT, hard, \**  
**acregmin=3, acregmax=60, acdirmin=30, acdirmax=60**

**actimeo** has no default; it sets **acregmin**, **acregmax**, **acdirmin** and **acdirmax**

Defaults for **rsize** and **wsiz** are set internally by the system kernel.

## UMOUNT OPTIONS

- h host** Unmount all filesystems listed in */etc/mstab* that are remote-mounted from *host*.
- t type** Unmount all filesystems listed in */etc/mstab* that are of a given *type*.
- a** Unmount all filesystems currently mounted (as listed in */etc/mstab*).
- v** Verbose. Display a message indicating each filesystem being unmounted.

## NFS FILESYSTEMS

### Background vs. Foreground

Filesystems mounted with the **bg** option indicate that **mount** is to retry in the background if the server's mount daemon (**mountd**(8C)) does not respond. **mount** retries the request up to the count specified in the **retry=*n*** option. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=*n*** tenths

of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When the number of retransmissions has reached the number specified in the `retrans=n` option, a filesystem mounted with the `soft` option returns an error on the request; one mounted with the `hard` option prints a warning message and continues to retry the request.

#### Read-Write vs. Read-Only

Filesystems that are mounted `rw` (read-write) should use the `hard` option.

#### Interrupting Processes With Pending NFS Requests

The `intr` option allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted filesystem.

#### Secure Filesystems

The `secure` option must be given if the server requires secure mounting for the filesystem.

#### File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting `actimeo=n` extends flush time by *n* seconds for both regular files and directories.

### SYSTEM V COMPATIBILITY

#### System V File-Creation Semantics

Ordinarily, when a file is created its GID is set to the effective GID of the calling process. This behavior may be overridden on a per-directory basis, by setting the set-GID bit of the parent directory; in this case, the GID is set to the GID of the parent directory (see `open(2V)` and `mkdir(2)`). Files created on filesystems that are mounted with the `grpuid` option will obey BSD semantics; that is, the GID is unconditionally inherited from that of the parent directory.

### EXAMPLES

|                                            |                                                   |
|--------------------------------------------|---------------------------------------------------|
| To mount a local disk:                     | <code>mount /dev/xy0g /usr</code>                 |
| To fake an entry for <code>nd</code> root: | <code>mount -ft 4.2 /dev/nd0 /</code>             |
| To mount all 4.2 filesystems:              | <code>mount -at 4.2</code>                        |
| To mount an NFS remote filesystem:         | <code>mount -t nfs serv:/usr/src /usr/src</code>  |
| To mount an NFS remote filesystem:         | <code>mount serv:/usr/src /usr/src</code>         |
| To hard mount an NFS remote filesystem:    | <code>mount -o hard serv:/usr/src /usr/src</code> |
| To save current mount state:               | <code>mount -p &gt; /etc/fstab</code>             |

### FILES

|                         |                                      |
|-------------------------|--------------------------------------|
| <code>/etc/mtab</code>  | table of mounted filesystems         |
| <code>/etc/fstab</code> | table of filesystems mounted at boot |

### WARNINGS

`mount` does not understand the mount order dependencies involved in loopback mounting. Loopback mounts may be dependent on two mounts having been previously performed, while `nfs` and `4.2` mounts are dependent only on a single previous mount. As a rule of thumb, place loopback mounts at the end of `/etc/fstab` file. See `lofs(4S)` for a complete description.

### SEE ALSO

`lofs(4S)`, `mountd(8C)`, `nfsd(8)`, `mkdir(2)`, `mount(2)`, `open(2V)`, `unmount(2)`, `fstab(5)`, `mtab(5)`,

### BUGS

Mounting filesystems full of garbage crashes the system.

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.



**NAME**

mountd, rpc.mountd – NFS mount request server

**SYNOPSIS**

/usr/etc/rpc.mountd [ -n ]

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**mountd** is an RPC server that answers file system mount requests. It reads the file */etc/xtab*, described in *exports(5)*, to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the *showmount(8)* command.

The **mountd** daemon is normally invoked by *rc(8)*.

**OPTIONS**

**-n** Do not check that the clients are root users. Though this option makes things slightly less secure, it does allow older versions (pre-3.0) of client NFS to work.

**FILES**

*/etc/xtab*

**SEE ALSO**

*exports(5)*, *rc(8)*, *showmount(8)*

## NAME

**setup\_client** – create or remove an NFS client

## SYNOPSIS

```
/usr/etc/install/script/setup_client op clientname yp_type swapsize rootpath swappath  
homepath execpath kvmpath arch
```

## DESCRIPTION

**setup\_client** adds an NFS client to a server, or removes one. It can only be run by the super-user. It is also used by **suninstall**(8).

The *op* argument indicates which operation to perform, either **add** or **remove**, to indicate whether to add or remove a client. *clientname* is the hostname of the client. *yp\_type* indicates the type of Yellow Pages server or service to provide to the client, if any; it can be one of **master**, **slave**, **client** or **none**. *swapsize* is the number of bytes reserved for client's swap file. *rootpath* is the pathname of the parent directory in which various client root directories reside; *rootpath/clientname* is the pathname of the client's root directory. *swappath* is the pathname of parent directory in which various client swap files reside; *swappath/clientname* is the pathname of the client's swap file. *homepath* is the pathname of the (parent) directory in which the various home directories are to reside; it is the pathname of the directory that the client is to mount as **/home**. *execpath* is the full pathname of the directory in which the executables for the kernel architecture specified by the *arch* argument. This is the directory that the client mounts as **/usr**. *kvmpath* is the full pathname of the directory in which the *kernel-specific* executables for the architecture specified by the *arch* argument reside. This is the directory that the client mounts as **/usr/kvm**. *arch* specifies the client's kernel architecture (for instance, **sun4**, **sun3x**...). See **arch**(1) for further explanation of "kernel architecture" and examples of valid kernel architectures. **setup\_client** with no arguments displays a usage message that includes the proper *arch* argument for each supported kernel architecture.

## USAGE

Before you add or remove a client, you must first make sure that the Internet and Ethernet addresses for *clientname* are listed in the YP hosts database (if the server is running the YP), or in the server's **/etc/hosts** and **/etc/ethers** databases, respectively (if otherwise). Then, run **setup\_client** with the **add** or **remove** operation. When adding a client, you must then bootstrap that client machine.

A server must support a client's specific kernel architecture before it can mount that client. The executable directory for that client's kernel architecture must be present on the server. If this directory is absent, an error results.

**setup\_client** updates the **/etc/bootparams** file. If the server is a YP master, it updates local YP database. It *does not* propagate the local update to other YP servers. To propagate the updates, use the following commands:

```
example# cd /var/yp  
example# make
```

If the server is running YP but is not a YP master, **setup\_client** issues a warning to indicate that the database is out of date.

When *arch* is given as **sun2**, **suninstall** issues a reminder to run the **/usr/etc/ndbootd** daemon for booting Sun-2 systems.

**setup\_client** creates *swappath/clientname* with the *size*, (number of bytes) you specify. You can append one of **K** or **k** to indicate kilobytes, **M** or **m** to indicate megabytes, or **B** or **b** to indicate 512-byte blocks, to *size*. Otherwise, *size* is taken to indicate an exact byte count.

**suninstall** updates the **/etc/exports** file to allow root access to each client's root file system. It exports the client's swap and dump partitions only to the client.

Note: The system administrator should verify that the **/etc/exports** file contains correct information, and that file systems are exported to the correct users and groups. Refer to **exportfs**(8) for details on exporting file systems.



**EXAMPLES**

This example shows how to add a Sun-4 system NFS client to a server.

```
example# setup_client add frodo client 16M /exports/roots /exports/swaps /exports/dumps /home \
/exports/execs/sun4/4.0 sun4
```

To remove this client, you would merely substitute **remove** for **add** in the above example.

**FILES**

|                  |                                              |
|------------------|----------------------------------------------|
| /etc/hosts       | hosts database                               |
| /etc/ethers      | database of hostnames and Ethernet addresses |
| /usr/etc/ndbootd | ND boot block server                         |
| /etc/bootparams  | boot parameter database                      |
| /etc/exports     | database of exported file systems            |

**SEE ALSO**

**exportfs(8), setup\_exec(8) suninstall(8)**

*Installing the SunOS*

**DIAGNOSTICS**

**incorrect number of arguments**

Check number and order of the arguments.

**must be run as root (super-user).**

**setup\_client** can only be used by root (super-user).

**invalid operation type "xx".**

Valid operations are **add** and **remove**.

**ATTENTION: xxxxxxxx -> boot.sun? not created.**

(Sun-3 systems only.) A symbolic link can not be created because the boot file does not exist.

**ATTENTION: xxxxxxxx.SUN? -> boot.sun? not created.**

(Other than Sun-3 systems.) A symbolic link can not be created because the boot file does not exist.

**ATTENTION: /usr/etc/ndbootd needs to be running on server before bringing up "client".**

The Sun-2 system boot daemon must be running in order to bootstrap a Sun-2 system.

## NAME

**setup\_exec** – install architecture-dependent executables on a heterogeneous file server

## SYNOPSIS

**/usr/etc/install/setup\_exec arch execpath kvmpath [-o]**

## DESCRIPTION

**setup\_exec** installs architecture-dependent executables from either a local tape drive or a remote host. It is used to convert a standalone system or homogeneous file server to a heterogeneous file server. **setup\_exec** is a forms-based utility that can be invoked directly, but it is also used by **suninstall(8)**. It can only be invoked by the super-user.

The *arch* argument specifies the kernel architecture to install (for instance, **sun4**, **sun3x...**). See **arch(1)** for further explanation of “kernel architecture” and examples of valid kernel architectures. When run with no arguments, **setup\_exec** displays a usage line that includes the proper format of the *arch* argument for each supported kernel architecture. *execpath* is the full pathname of the directory in which to install the executables. When **setup\_exec** is done, the *execpath* directory is ready to mount as **/usr** by the heterogeneous server’s NFS clients of the indicated *arch*. *kvmpath* is the full pathname of the directory in which to install the *kernel-specific* executables (**libkvm**, **pstat(8)**, **ps(1)**, etc.). When **setup\_exec** is done, the *kvmpath* directory is ready to mount as **/usr/kvm** by the heterogeneous server’s NFS clients of the indicated *arch*.

**setup\_exec** also updates the **/etc/exports** file (see **exportfs(8)**) to export the executable directories it has installed. A system administrator should verify this file to make sure that the directory has been exported to the correct groups.

## OPTIONS

**-o** Override the software duplication prevention mechanism. Allows the user to reload a software category that is already installed.

## EXAMPLE

This example shows how to install a directory of executables for Sun-4 system clients.

```
example# setup_exec sun4 /export/exec/kvm/sun4
```

## FILES

|                                                |                                                                      |
|------------------------------------------------|----------------------------------------------------------------------|
| <b>/etc/hosts</b>                              | hosts database                                                       |
| <b>/etc/ethers</b>                             | database of hostnames and Ethernet addresses                         |
| <b>/etc/exports</b>                            | database of exported file systems                                    |
| <b>/usr/etc/install/files/extractlist.arch</b> | record of extracted categories for the indicated kernel architecture |
| <b>/usr/etc/install/save</b>                   | history of software installation                                     |

## SEE ALSO

**exportfs(8)**, **setup\_client(8)**, **suninstall(8)**

*Installing the SunOS*

## DIAGNOSTICS

**incorrect number of arguments**

Check the number and the order of arguments.

**invalid architecture type “arch”.**

An unsupported value for *arch* was supplied.

**invalid tape drive type “drive”.**

Valid tape drive types are local and remote.

**invalid tape type “tape”.**

Valid tape types are ar, st, mt, and xt.

**can’t reach tapehost “tapehost”.**

The IP address of *tapehost* is not in the hosts database, that is, the hosts YP database if the Yellow Pages are running, or the **/etc/hosts** file otherwise.





**Load release tape *n***

Mount the release tape specified on the screen and type RETURN to continue.



**NAME**

**showmount** – show all remote mounts

**SYNOPSIS**

**/usr/etc/showmount** [ **-ade** ] [ **host** ]

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**showmount** lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the **mountd(8C)** server on *host*, and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the value returned by **hostname(1)**.

**OPTIONS**

**-a**      Print all remote mounts in the format

*hostname:directory*

where **hostname** is the name of the client, and **directory** is the root of the file system that has been mounted.

**-d**      List directories that have been remotely mounted by clients.

**-e**      Print the list of exported file systems.

**FILES**

**/etc/rmtab**

**SEE ALSO**

**hostname(1)**, **exports(5)**, **exports(5)**, **mountd(8C)**

**BUGS**

If a client crashes, its entry will not be removed from the list until it reboots and executes '**umount -a**'.

**NAME**

**sunupgrade** – upgrade the Sun Operating System

**SYNOPSIS**

**/usr/etc/sunupgrade** [ -l ] [ -d ] [ -n ]

**AVAILABILITY**

This command is available on sun2, sun3 and sun4 application architectures running SunOS version 4.0 only. Refer to *Installing the SunOS 4.0.3* for more information.

**DESCRIPTION**

**sunupgrade** is an interactive utility that is used to upgrade the Sun Operating System (SunOS) to a higher revision level on sun2, sun3, and sun4 application architectures. The current SunOS level must be at least SunOS 4.0.

**sunupgrade** lets you upgrade any system configuration. The following are the valid configuration types:

- Standalone
- Homogeneous server
- Heterogeneous server
- Dataless clients
- Diskless clients

Both local and remote installation modes are supported.

**sunupgrade** overlays the newer executable files on top of existing ones. For files in and if the new file names and old file names conflict and the files are not identical in content, then the new file is installed with a trailing suffix that is the release name. The differences between the old and the new versions of the files must be resolved by the user. All such files are logged in the log file **/usr/etc/upgrade/save/special\_files** for servers, standalone systems, or in **/usr/etc/upgrade/save/clientname.special\_files** for diskless clients. For dataless clients, they are shared under **/home/upgrade/special\_files**.

After **sunupgrade** completes execution you must come up in single-user mode, inspect all special files and propagate your older administrative files to the newer ones and rename them without the suffixes.

**OPTIONS**

- l Create log of all files extracted and overlaid. Performance will deteriorate slightly. Log files are saved in **/usr/etc/upgrade/save**. Special files are called
- d Work in debugging mode. Not recommended for normal operation.
- n Switch off "no-rewind" operation. The no-rewind operation available only on systems running SunOS 4.0.2 or 4.0.3.

**FILES**

**/usr/etc/upgrade/EXCLUDELIST**  
**/usr/etc/upgrade/README**  
**/usr/etc/upgrade/checksums**  
**/usr/etc/upgrade/chk\_ok**  
**/usr/etc/upgrade/chk\_release**  
**/usr/etc/upgrade/chkextract**  
**/usr/etc/upgrade/config\_host**  
**/usr/etc/upgrade/extract**  
**/usr/etc/upgrade/extract\_client**  
**/usr/etc/upgrade/extract\_clntroot**  
**/usr/etc/upgrade/extract\_stand**  
**/usr/etc/upgrade/include**  
**/usr/etc/upgrade/includefile**  
**/usr/etc/upgrade/get\_arch**  
**/usr/etc/upgrade/get\_clientinfo**  
**/usr/etc/upgrade/get\_machtype**



`/usr/etc/upgrade/get_tapeinfo`  
`/usr/etc/upgrade/get_toc`  
`/usr/etc/upgrade/get_upgradeinfo`  
`/usr/etc/upgrade/mop_up`  
`/usr/etc/upgrade/mount_ufs`  
`/usr/etc/upgrade/mount_usr`  
`/usr/etc/upgrade/start_log`  
`/usr/etc/upgrade/setup_kvm`  
`/usr/etc/upgrade/small_kernel_files`  
`/usr/etc/upgrade/sun2_cp_share`  
`/usr/etc/upgrade/sun2_ln_exec`  
`/usr/etc/upgrade/sunupgrade`  
`/usr/etc/upgrade/tar_clntroot`  
`/usr/etc/upgrade/verify_clntpart`  
`/usr/etc/upgrade/xdrtoc`

**SEE ALSO**

`suninstall(8)`

*Installing the SunOS 4.0.3*

**NAME**

syslogd – log system messages

**SYNOPSIS**

**/usr/etc/syslogd** [ **-d** ] [ **-f***configfile* ] [ **-m** *interval* ]

**DESCRIPTION**

**syslogd** reads and forwards system messages to the appropriate log files and/or users, depending upon the priority of a message and the system facility from which it originates. The configuration file **/etc/syslog.conf** (see **syslog.conf(5)**) controls where messages are forwarded. **syslogd** logs a mark (timestamp) message every *interval* minutes (default 20) at priority LOG\_INFO to the facility whose name is given as **mark** in the **syslog.conf** file.

A system message consists of a single line of text, which may be prefixed with a priority code number enclosed in angle-brackets (<>); priorities are defined in **sys/syslog.h**.

**syslogd** reads from the AF\_UNIX address family socket **/dev/log**, from an Internet address family socket specified in **/etc/services**, and from the special device **/dev/klog** (for kernel messages).

**syslogd** reads the configuration file when it starts up, and again whenever it receives a HUP signal, at which time it also closes all files it has open, re-reads its configuration file, and then opens only the log files that are listed in that file. **syslogd** exits when it receives a TERM signal.

As it starts up, **syslogd** creates the file **/etc/syslog.pid**, if possible, containing its process ID (PID).

**Sun386i DESCRIPTION**

**syslogd** translates messages using the databases specified on an optional line in the **syslog.conf** as indicated with a **translate** entry.

The format of these databases is described in **translate(5)**.

**OPTIONS**

|                             |                                                         |
|-----------------------------|---------------------------------------------------------|
| <b>-d</b>                   | Turn on debugging.                                      |
| <b>-f</b> <i>configfile</i> | Specify an alternate configuration file.                |
| <b>-m</b> <i>interval</i>   | Specify an interval, in minutes, between mark messages. |

**FILES**

|                         |                                            |
|-------------------------|--------------------------------------------|
| <b>/etc/syslog.conf</b> | configuration file                         |
| <b>/etc/syslog.pid</b>  | process ID                                 |
| <b>/dev/log</b>         | AF_UNIX address family datagram log socket |
| <b>/dev/klog</b>        | kernel log device                          |
| <b>/etc/services</b>    | network services database                  |

**SEE ALSO**

**logger(1)**, **syslog(3)**, **syslog.conf(5)**, **translate(5)**





**NAME**

sundiag – system diagnostics

**SYNOPSIS**

```
/usr/diag/sundiag/sundiag [ -Cmptv ] [ -h remotehost ] [ -o saved_options_file ]
    [ generic_tool_arguments ]
```

**AVAILABILITY**

This program is available with the *User Diagnostics* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**sundiag** is a diagnostic facility that tests the functionality of the operating system and reports its findings. It can also be used to report the hardware configuration as detected by the system.

You must be root to use **sundiag**.

When run on the console monitor, **sundiag** takes full advantage of the *SunView 1* windowing environment. There are four subwindows: a control panel for displaying the discovered hardware configuration and manipulating of the numerous test parameters and options, a test status panel which shows the test results, a console window which is used to display messages, and a performance monitor. There are also some popup frames, including a text frame for viewing **sundiag** and system log files.

When executed from a terminal, **sundiag** uses **curses(3X)** to simulate each subwindow on the screen.

**sundiag** consists of **sundiag**, along with several binary modules and executable files containing the actual test code, all of which reside in **/usr/diag/sundiag**.

**OPTIONS**

- C** Redirect the console output from any existing console window to the **sundiag** console subwindow. This option does not work when running on a terminal or tty emulator.
- m** Create a device file for all devices found during the kernel probe. **sundiag** uses the same major/minor device numbers and permissions declared in **/dev/MAKEDEV**.
- p** Ignore the kernel probe for devices, especially when running user-defined tests found in the **.user-test** file.
- t** Run **sundiag** on a terminal. Incompatible with **-C**.
- v** Suppress the **sundiag** start-up messages so that they do not interfere with the display when SunView windows come up. This argument may be used in the **.sunview** file.
- h remotehost**  
Use this option to invoke **sundiag** when using the SunView-based Remote Interface. Refer to the *Sundiag User's Guide* for information on this interface.
- o saved\_options\_file**  
Use the **saved\_options\_file** to restore options. The default option file is **.sundiag**. **.sundiag** is used if the **-o** option is not used and if the default file exists.

*generic\_tool\_arguments*

Refer to **sunview(1)** for examples of generic tool arguments that may be used with **sundiag**.

**FILES**

|                                             |                                    |
|---------------------------------------------|------------------------------------|
| <b>/var/adm/sundiaglog/options/.sundiag</b> | start-up option file               |
| <b>/usr/diag/sundiag.usertest</b>           | user-defined test description file |

**SEE ALSO**

**sunview(1)**, **curses(3X)**

*Installing the SunOS*  
*Sundiag User's Guide*



**NAME**

talkd, in.talkd – server for talk program

**SYNOPSIS**

/usr/etc/in.talkd

**DESCRIPTION**

**talkd** is a server used by the **talk(1)** program. It listens at the udp port indicated in the “talk” service description; see **services(5)**. The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

**SEE ALSO**

**talk(1)**, **services(5)**, **inetd(8C)**

**BUGS**

The protocol is architecture dependent, and can not be relied upon to work between Sun systems and other machines.

---

Notes



---

## Notes

---

## Notes



---

Notes

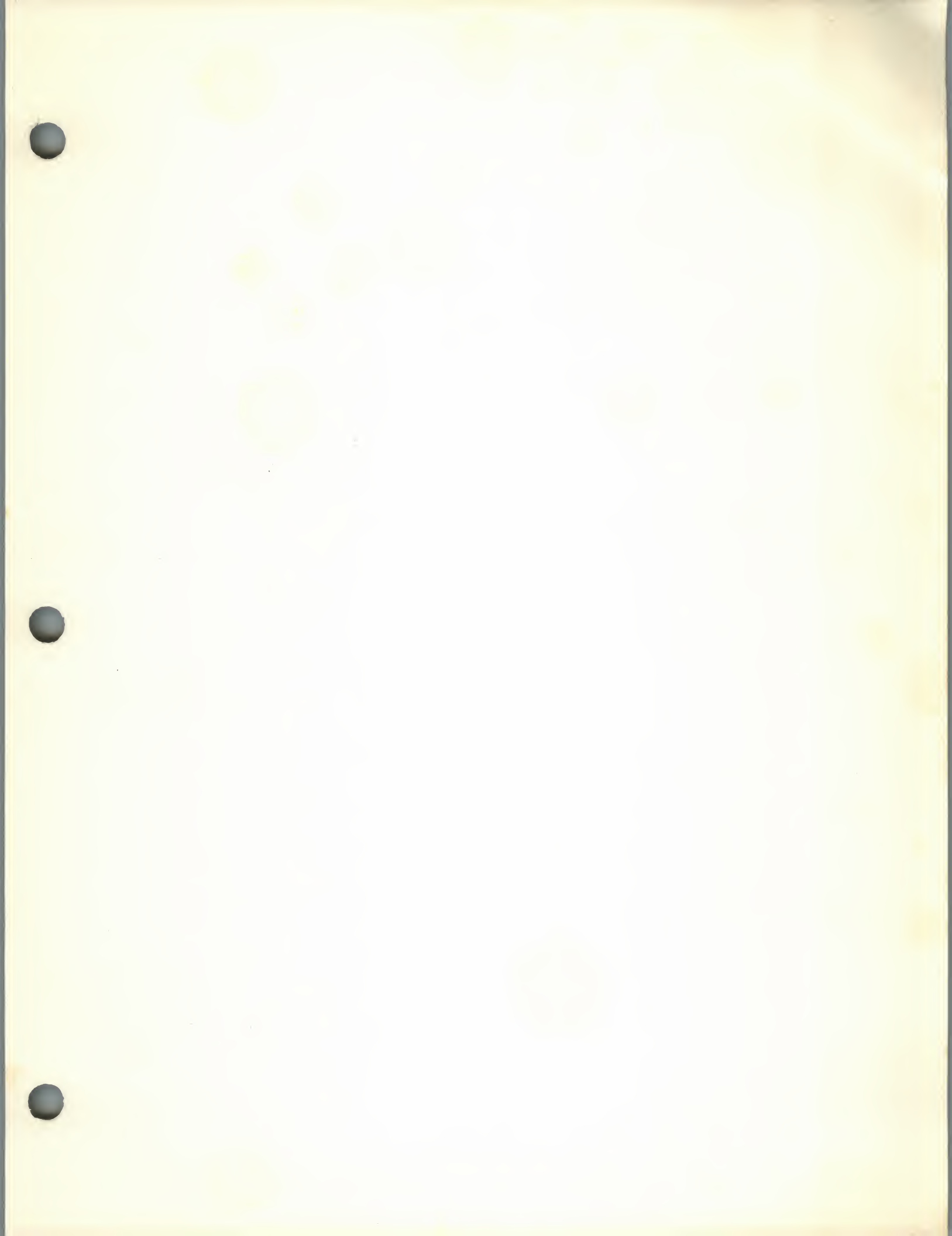
---

Notes



---

Notes





## Systems for Open Computing™

### Corporate Headquarters

Sun Microsystems, Inc.  
2550 Garcia Avenue  
Mountain View, CA 94043  
415 960-1300  
TLX 37-29639

### For U.S. Sales Office

locations, call:  
800 821-4643  
In CA: 800 821-4642

### European Headquarters

Sun Microsystems Europe, Inc.  
Bagshot Manor, Green Lane  
Bagshot, Surrey GU19 5NL  
England  
0276 51440  
TLX 859017

Australia: (02) 413 2666

Canada: 416 477-6745

France: (1) 40 94 80 00

Germany: (089) 95094-0

Hong Kong: 852 5-8651688

Italy: (39) 6056337

Japan: (03) 221-7021

Korea: 2-7802255

Nordic Countries: + 46 (0)8 7647810

PRC: 1-8315568

Singapore: 224 3388

Spain: (1) 2532003

Switzerland: (1) 8289555

The Netherlands: 3133501234

Taiwan: 2-7213257

UK: 0276 62111

Europe, Middle East, and Africa,  
call European Headquarters:  
0276 51440

Elsewhere in the world,  
call Corporate Headquarters:  
415 960-1300  
Intercontinental Sales



---

# Network Software Environment<sup>TM</sup> Reference Manual



Part No: 800-3174-10  
Revision A of 7 October 1988



## Credits and Trademarks

NSE is a trademark of Sun Microsystems, Inc.

Sun Microsystems and Sun Workstation are registered trademarks of Sun Microsystems, Inc.

SunOS, Sun Microsystems, SunView, SunWindows, Sun-2, Sun-3, Sun-4, and Sun386i are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of A T & T .

Copyright © 1988, Sun Microsystems, Inc. All Rights Reserved. No part of this work covered by copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without permission of the copyright owner.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

---

# Contents

|                                         |      |
|-----------------------------------------|------|
| Preface .....                           | xix  |
| The Audience .....                      | xix  |
| Software Development Organization ..... | xix  |
| Release Management .....                | xx   |
| Software Development Management .....   | xx   |
| Program Development .....               | xx   |
| Documentation Writing .....             | xx   |
| Other Development .....                 | xxi  |
| Customer Support .....                  | xxi  |
| System Administration .....             | xxi  |
| Typographical Conventions .....         | xxi  |
| Documentation Set .....                 | xxii |
| The Browser .....                       | 1    |
| 1.1 Starting Up the Browser .....       | 2    |
| 1.2 Browser Window Header .....         | 2    |
| 1.3 Browser Subwindows .....            | 4    |
| Legend Subwindow .....                  | 4    |
| Display Subwindow .....                 | 5    |
| Log Subwindow .....                     | 6    |
| 1.4 Browser Symbols .....               | 7    |
| Environment Symbol .....                | 8    |



|     |                                         |    |
|-----|-----------------------------------------|----|
|     | Component Symbol .....                  | 9  |
|     | make Target Symbol .....                | 10 |
|     | File Symbol .....                       | 11 |
|     | Link Database Symbol .....              | 12 |
|     | Variant Symbol .....                    | 13 |
|     | Symbol Summary .....                    | 14 |
| 1.5 | Browser Menus .....                     | 15 |
|     | Selection Menus .....                   | 17 |
|     | Window Menus .....                      | 35 |
|     | Frame Menu .....                        | 40 |
| 1.6 | Dialog Boxes and Popup Windows .....    | 42 |
|     | Dialog Boxes .....                      | 42 |
|     | Confirmation Windows .....              | 46 |
|     | History Windows .....                   | 47 |
|     | Output Windows .....                    | 48 |
| 1.7 | Browser Operations and Menu Items ..... | 49 |
|     | Acquire .....                           | 50 |
|     | Add Member... .....                     | 53 |
|     | All Components .....                    | 55 |
|     | Append .....                            | 56 |
|     | Checkin... .....                        | 58 |
|     | Checkout .....                          | 60 |
|     | Child Envs .....                        | 62 |
|     | Clean Up .....                          | 63 |
|     | Contents .....                          | 64 |
|     | Dbxtool .....                           | 65 |
|     | Delete => .....                         | 67 |
|     | Descriptors .....                       | 68 |
|     | Diffs => .....                          | 69 |

|                            |     |
|----------------------------|-----|
| Dismiss.....               | 71  |
| (Editor).....              | 72  |
| Emacs.....                 | 73  |
| Get Version... ..          | 75  |
| History .....              | 76  |
| Kill .....                 | 78  |
| Latest.....                | 79  |
| List =>.....               | 80  |
| List History .....         | 89  |
| List My Envs.....          | 90  |
| List Other's Envs... ..    | 91  |
| List Public Envs .....     | 92  |
| List Source Files .....    | 93  |
| Lock .....                 | 94  |
| Make =>.....               | 95  |
| Notifications.....         | 97  |
| Other Target... ..         | 98  |
| Parent Envs .....          | 99  |
| Preserve.....              | 100 |
| Previous Queries =>.....   | 101 |
| Reconcile => .....         | 102 |
| Recover =>.....            | 103 |
| Register Notification..... | 104 |
| Remove .....               | 106 |
| Rename... ..               | 107 |
| Resolve => .....           | 111 |
| Resync => .....            | 112 |
| Root Component .....       | 113 |
| Shelltool .....            | 114 |



|     |                                             |     |
|-----|---------------------------------------------|-----|
|     | Show Output .....                           | 115 |
|     | Textedit .....                              | 116 |
|     | Two Versions.....                           | 118 |
|     | Uncheckout .....                            | 119 |
|     | Unlock.....                                 | 120 |
|     | Unregister.....                             | 121 |
|     | Variants .....                              | 122 |
|     | Version.....                                | 123 |
|     | Vi .....                                    | 124 |
|     | You Specify... ..                           | 125 |
| 1.8 | Using A Text Editor with the Browser .....  | 126 |
|     | Resolving Objects .....                     | 131 |
| 2.1 | Resolve.....                                | 131 |
| 2.2 | Merging With Windows .....                  | 132 |
| 2.3 | Auto Merging.....                           | 132 |
| 2.4 | A Sample Resolve.....                       | 133 |
|     | fileresolve.....                            | 133 |
|     | dbresolve.....                              | 135 |
|     | Links and the Link Service .....            | 139 |
| 3.1 | Introduction.....                           | 139 |
| 3.2 | Link Databases.....                         | 140 |
| 3.3 | The Link Service Operating Environment..... | 140 |
| 3.4 | User Interface.....                         | 141 |
|     | Starting up a Link Service Window.....      | 141 |
|     | Link Service Window Header Information..... | 141 |
|     | Link Service Popup Menus.....               | 142 |
| 3.5 | Link Service Operations .....               | 143 |
|     | Cancel .....                                | 143 |
|     | Confirm All.....                            | 144 |

|                                                     |     |
|-----------------------------------------------------|-----|
| Create .....                                        | 144 |
| Database .....                                      | 144 |
| Delete .....                                        | 145 |
| Display .....                                       | 145 |
| End Link .....                                      | 146 |
| Set .....                                           | 147 |
| Start Link .....                                    | 147 |
| Command-Line Interface .....                        | 151 |
| 4.1 Command Prompt for Activated Environments ..... | 151 |
| 4.2 Command-Line Interface Example .....            | 152 |
| Setting Up a Command-Line Interface Example .....   | 152 |
| Running a Command-Line Interface Example .....      | 153 |
| 4.3 Command Summaries .....                         | 163 |
| acquire .....                                       | 164 |
| activate .....                                      | 168 |
| append .....                                        | 170 |
| bootstrap .....                                     | 173 |
| dbresolve .....                                     | 175 |
| describe_comps .....                                | 179 |
| fileresolve .....                                   | 181 |
| linksvc .....                                       | 186 |
| lvcs .....                                          | 187 |
| merged .....                                        | 189 |
| nsebrowse .....                                     | 193 |
| nsecomp .....                                       | 195 |
| nsecontrolpt .....                                  | 199 |
| nseenv .....                                        | 201 |
| nseexecset .....                                    | 213 |
| nsegc .....                                         | 218 |



|                                 |     |
|---------------------------------|-----|
| nseivp .....                    | 220 |
| nsemark .....                   | 221 |
| nsenotify .....                 | 222 |
| nsemlink .....                  | 226 |
| nsevar .....                    | 229 |
| preserve .....                  | 233 |
| reconcile .....                 | 236 |
| resolve .....                   | 241 |
| resync .....                    | 243 |
| vcs .....                       | 246 |
| NSE Naming Specifications ..... | 255 |
| 5.1 Environment Names .....     | 255 |
| 5.2 Variant Names .....         | 256 |
| 5.3 Component Names .....       | 256 |
| 5.4 Target Names .....          | 257 |
| 5.5 Filenames .....             | 257 |
| 5.6 Link Database Names .....   | 257 |
| Glossary .....                  | 261 |

---

## Tables

|           |                                            |     |
|-----------|--------------------------------------------|-----|
| Table 1-1 | Browser Symbols .....                      | 14  |
| Table 1-2 | Browser: File Editor Operations .....      | 126 |
| Table 4-1 | Notification Events .....                  | 222 |
| Table 4-2 | Notification Event-Specific Arguments..... | 222 |





---

## Figures

|             |                                                            |    |
|-------------|------------------------------------------------------------|----|
| Figure 1-1  | The Network Software Environment Browser .....             | 1  |
| Figure 1-2  | Browser: Start Up .....                                    | 2  |
| Figure 1-3  | Browser: Window Header at Start Up .....                   | 2  |
| Figure 1-4  | Browser: Legend Subwindow .....                            | 4  |
| Figure 1-5  | Browser: Legend Subwindow – Location in<br>Hierarchy ..... | 4  |
| Figure 1-6  | Browser: Display Subwindow .....                           | 5  |
| Figure 1-7  | Browser: Log Subwindow .....                               | 6  |
| Figure 1-8  | Browser: Environment Symbol and Legend .....               | 8  |
| Figure 1-9  | Browser: Environment Symbol in Display .....               | 8  |
| Figure 1-10 | Browser: Component Symbol and Legend .....                 | 9  |
| Figure 1-11 | Browser: Component Symbol in Display .....                 | 9  |
| Figure 1-12 | Browser: make Target Symbol and Legend .....               | 10 |
| Figure 1-13 | Browser: make Target Symbol in Display .....               | 10 |
| Figure 1-14 | Browser: File Symbol in Display .....                      | 11 |
| Figure 1-15 | Browser: Link Database Symbol in Display .....             | 12 |
| Figure 1-16 | Browser: Variant Symbol in Display .....                   | 13 |
| Figure 1-17 | Browser: Queries and How They Interrelate .....            | 16 |
| Figure 1-18 | Browser: Environment Menu .....                            | 18 |



|             |                                                     |    |
|-------------|-----------------------------------------------------|----|
| Figure 1-19 | Browser: Environment Menu With Items Expanded.....  | 18 |
| Figure 1-20 | Browser: Component Menu.....                        | 21 |
| Figure 1-21 | Browser: Component Menu With Items Expanded ....    | 21 |
| Figure 1-22 | Browser: Target Menu .....                          | 23 |
| Figure 1-23 | Browser: Target Menu With Item Expanded.....        | 23 |
| Figure 1-24 | Browser: File Menu .....                            | 25 |
| Figure 1-25 | Browser: File Menu With Item Expanded .....         | 25 |
| Figure 1-26 | Browser: Link Database Menu .....                   | 28 |
| Figure 1-27 | Browser: Variant Menu.....                          | 30 |
| Figure 1-28 | Browser: Variant Menu With Items Expanded.....      | 30 |
| Figure 1-29 | Browser: Notification Menu .....                    | 32 |
| Figure 1-30 | Browser: Log Item Menu.....                         | 33 |
| Figure 1-31 | Browser: Summary of Selection Menus .....           | 34 |
| Figure 1-32 | Browser: Legend Menu.....                           | 35 |
| Figure 1-33 | Browser: Display Menu .....                         | 36 |
| Figure 1-34 | Browser: Log Menu .....                             | 38 |
| Figure 1-35 | Browser: Summary of Window Menus .....              | 39 |
| Figure 1-36 | Browser: Frame Menu .....                           | 40 |
| Figure 1-37 | Browser: Frame Menu With Items Expanded .....       | 41 |
| Figure 1-38 | Browser: Dialog Box .....                           | 42 |
| Figure 1-39 | Browser: Dialog Box, Parameters Specification ..... | 43 |
| Figure 1-40 | Browser: Dialog Box, Options Specification.....     | 44 |
| Figure 1-41 | Browser: Dialog Box, Comments Specification.....    | 45 |
| Figure 1-42 | Browser: Confirmation Window.....                   | 46 |

|             |                                                             |    |
|-------------|-------------------------------------------------------------|----|
| Figure 1-43 | Browser: History Window .....                               | 47 |
| Figure 1-44 | Browser: Output Window .....                                | 48 |
| Figure 1-45 | Browser: Acquire... on Component Menu .....                 | 50 |
| Figure 1-46 | Browser: Acquire... on Component Menu,<br>Dialog Box .....  | 50 |
| Figure 1-47 | Browser: Acquire... on Display Menu .....                   | 51 |
| Figure 1-48 | Browser: Acquire... on Display Menu,<br>Dialog Box .....    | 52 |
| Figure 1-49 | Browser: Add Member... .....                                | 53 |
| Figure 1-50 | Browser: The Add Member... Dialog Box.....                  | 53 |
| Figure 1-51 | Browser: Add Member... .....                                | 54 |
| Figure 1-52 | Browser: All Components .....                               | 55 |
| Figure 1-53 | Browser: Append... on Environment Menu .....                | 56 |
| Figure 1-54 | Browser: Append... on Environment Menu,<br>Dialog Box ..... | 57 |
| Figure 1-55 | Browser: Checkin... on File Menu .....                      | 58 |
| Figure 1-56 | Browser: Checkin... on File Menu, Dialog Box .....          | 58 |
| Figure 1-57 | Browser: Checkin... on File Menu,<br>After Operation .....  | 59 |
| Figure 1-58 | Browser: Checkout on File Menu .....                        | 60 |
| Figure 1-59 | Browser: Checkout on File Menu,<br>After Operation .....    | 61 |
| Figure 1-60 | Browser: Child Envs .....                                   | 62 |
| Figure 1-61 | Browser: Clean Up.....                                      | 63 |
| Figure 1-62 | Browser: Contents.....                                      | 64 |
| Figure 1-63 | Browser: Dbxtool.....                                       | 65 |
| Figure 1-64 | Browser: Dbxtool, After Operation .....                     | 66 |



|             |                                                              |    |
|-------------|--------------------------------------------------------------|----|
| Figure 1-65 | Browser: Delete => .....                                     | 67 |
| Figure 1-66 | Browser: Delete => Options..., Popup Window .....            | 67 |
| Figure 1-67 | Browser: Descriptors .....                                   | 68 |
| Figure 1-68 | Browser: Descriptors, Descriptive Information .....          | 68 |
| Figure 1-69 | Browser: Diffs => .....                                      | 69 |
| Figure 1-70 | Browser: Diffs => Latest and filemerge .....                 | 69 |
| Figure 1-71 | Browser: Emacs .....                                         | 73 |
| Figure 1-72 | Browser: Emacs, Popup Window .....                           | 74 |
| Figure 1-73 | Browser: Get Version... on File Menu .....                   | 75 |
| Figure 1-74 | Browser: Get Version... on File Menu,<br>Dialog Box .....    | 75 |
| Figure 1-75 | Browser: History on Environment Menu .....                   | 76 |
| Figure 1-76 | Browser: History on Environment Menu,<br>Popup Window .....  | 77 |
| Figure 1-77 | Browser: Kill .....                                          | 78 |
| Figure 1-78 | Browser: Latest .....                                        | 79 |
| Figure 1-79 | Browser: List => on Environment Menu .....                   | 80 |
| Figure 1-80 | Browser: List => All Components on<br>Environment Menu ..... | 81 |
| Figure 1-81 | Browser: List => Root Component on<br>Environment Menu ..... | 81 |
| Figure 1-82 | Browser: List => History on Environment Menu .....           | 82 |
| Figure 1-83 | Browser: List => Child Envs on Environment<br>Menu .....     | 82 |
| Figure 1-84 | Browser: List => Parent Envs on Environment<br>Menu .....    | 83 |
| Figure 1-85 | Browser: List => Notifications on Environment<br>Menu .....  | 83 |

|              |                                                               |     |
|--------------|---------------------------------------------------------------|-----|
| Figure 1-86  | Browser: List => Variants on Environment Menu .....           | 84  |
| Figure 1-87  | Browser: List => on Component Menu .....                      | 85  |
| Figure 1-88  | Browser: List => Contents on Component Menu .....             | 85  |
| Figure 1-89  | Browser: List => History on Component Menu .....              | 86  |
| Figure 1-90  | Browser: List => on Variant Menu .....                        | 87  |
| Figure 1-91  | Browser: List => Descriptors on Variant Menu .....            | 88  |
| Figure 1-92  | Browser: List History .....                                   | 89  |
| Figure 1-93  | Browser: List My Envs on Display Menu .....                   | 90  |
| Figure 1-94  | Browser: List Other's Envs... on Display Menu .....           | 91  |
| Figure 1-95  | Browser: List Other's Envs..., Display Menu, Dialog Box ..... | 91  |
| Figure 1-96  | Browser: List Public Envs on Display Menu .....               | 92  |
| Figure 1-97  | Browser: List Source Files .....                              | 93  |
| Figure 1-98  | Browser: List Source Files, After Operation .....             | 93  |
| Figure 1-99  | Browser: Lock .....                                           | 94  |
| Figure 1-100 | Browser: Make => .....                                        | 95  |
| Figure 1-101 | Browser: Make => Options..., Popup Window .....               | 95  |
| Figure 1-102 | Browser: Make => Other Target..., Dialog Box .....            | 96  |
| Figure 1-103 | Browser: Notifications .....                                  | 97  |
| Figure 1-104 | Browser: Other Target... .....                                | 98  |
| Figure 1-105 | Browser: Parent Envs .....                                    | 99  |
| Figure 1-106 | Browser: Preserve .....                                       | 100 |
| Figure 1-107 | Browser: Preserve..., Dialog Box .....                        | 100 |
| Figure 1-108 | Browser: Previous Queries => .....                            | 101 |
| Figure 1-109 | Browser: Reconcile => .....                                   | 102 |



|              |                                                             |     |
|--------------|-------------------------------------------------------------|-----|
| Figure 1-110 | Browser: Reconcile => Options...,<br>Popup Window .....     | 102 |
| Figure 1-111 | Browser: Recover => .....                                   | 103 |
| Figure 1-112 | Browser: Register Notification... ..                        | 104 |
| Figure 1-113 | Browser: Register Notification..., Dialog Box.....          | 104 |
| Figure 1-114 | Browser: Remove on Component Menu.....                      | 106 |
| Figure 1-115 | Browser: Rename... on Environment Menu.....                 | 107 |
| Figure 1-116 | Browser: Rename... on Environment Menu,<br>Dialog Box ..... | 107 |
| Figure 1-117 | Browser: Rename... on Component Menu.....                   | 108 |
| Figure 1-118 | Browser: Rename... on File Menu .....                       | 108 |
| Figure 1-119 | Browser: Rename... on Link Database Menu .....              | 109 |
| Figure 1-120 | Browser: Resolve => .....                                   | 111 |
| Figure 1-121 | Browser: Resolve => Options...,<br>Popup Window .....       | 111 |
| Figure 1-122 | Browser: Resync =>.....                                     | 112 |
| Figure 1-123 | Browser: Resync => Options...,<br>Popup Window .....        | 112 |
| Figure 1-124 | Browser: Root Component on Environment<br>Menu .....        | 113 |
| Figure 1-125 | Browser: Shelltool on Environment Menu .....                | 114 |
| Figure 1-126 | Browser: Show Output.....                                   | 115 |
| Figure 1-127 | Browser: Textedit.....                                      | 116 |
| Figure 1-128 | Browser: Textedit, Popup Window.....                        | 117 |
| Figure 1-129 | Browser: Two Versions... ..                                 | 118 |
| Figure 1-130 | Browser: Uncheckout on File Menu .....                      | 119 |
| Figure 1-131 | Browser: Unlock .....                                       | 120 |

|              |                                                              |     |
|--------------|--------------------------------------------------------------|-----|
| Figure 1-132 | Browser: Variants .....                                      | 122 |
| Figure 1-133 | Browser: Version.....                                        | 123 |
| Figure 1-134 | Browser: Vi.....                                             | 124 |
| Figure 1-135 | Browser: Vi, Popup Window .....                              | 124 |
| Figure 1-136 | Browser: You Specify.....                                    | 125 |
| Figure 1-137 | Browser: You Specify..., Popup Window.....                   | 125 |
| Figure 1-138 | Browser: File Editor, Symbolic Link.....                     | 126 |
| Figure 2-1   | fileresolve.....                                             | 133 |
| Figure 2-2   | dbresolve .....                                              | 135 |
| Figure 3-1   | Starting Up a Link Service Window .....                      | 141 |
| Figure 3-2   | Link Service Window Header.....                              | 141 |
| Figure 3-3   | Link Service Window: Frame Menu.....                         | 142 |
| Figure 3-4   | Link Service Icon: Frame Menu .....                          | 142 |
| Figure 3-5   | Link Service Database Popup Window:<br>Pull-Right Menu ..... | 143 |
| Figure 4-1   | Command Prompt Insert for .cshrc File.....                   | 151 |
| Figure 4-2   | Checking Your Environment List.....                          | 152 |
| Figure 4-3   | Setting Up a Command-Line Interface Example.....             | 153 |
| Figure 4-4   | Acquire Component and Activate Child<br>Environment.....     | 154 |
| Figure 4-5   | Checking Out a File from the Version Control<br>System.....  | 154 |
| Figure 4-6   | Line to Add to reasons.h File.....                           | 155 |
| Figure 4-7   | Listing the reasons.h File .....                             | 155 |
| Figure 4-8   | Checking In a File to the Version Control<br>System.....     | 155 |
| Figure 4-9   | Building the Changed Component.....                          | 155 |



|             |                                                                      |     |
|-------------|----------------------------------------------------------------------|-----|
| Figure 4-10 | Testing the Changed Component.....                                   | 156 |
| Figure 4-11 | Attempt reconcile of Changed Component:<br>No Changes in Parent..... | 156 |
| Figure 4-12 | Simulating a Change in the Parent Environment.....                   | 157 |
| Figure 4-13 | Checking Out a File from the Version Control<br>System.....          | 158 |
| Figure 4-14 | Line to Add to reasons.h File.....                                   | 158 |
| Figure 4-15 | Listing the reasons.h File .....                                     | 158 |
| Figure 4-16 | Checking In a File to the Version Control<br>System.....             | 159 |
| Figure 4-17 | Building the Changed Component.....                                  | 159 |
| Figure 4-18 | Testing the Changed Component.....                                   | 159 |
| Figure 4-19 | Attempt reconcile of Changed Component:<br>Changes in Parent .....   | 160 |
| Figure 4-20 | Performing a resolve .....                                           | 160 |
| Figure 4-21 | Listing Resolution Files .....                                       | 161 |
| Figure 4-22 | Marking File as Merged.....                                          | 161 |
| Figure 4-23 | Garbage Collection of Unmarked Environments.....                     | 219 |

---

# Preface

## The Audience

The audience of this book is diverse because the Network Software Environment (NSE) is useful to many people within a software development organization. People who use the Network Software Environment product include:

- Overall Software Development Strategists
- Release Managers
- Software Development Managers
- Software Engineers, or Programmers
- Documentation Writers
- Other Developers
- Customer Support Personnel
- System Administrators

To use this product, you should have a general knowledge of the SunOS operating system, which you can obtain from the Sun Beginner's Guides, especially the *Getting Started with UNIX: Beginner's Guide*.

If you are installing this product, or if you are interested only in system administration of the Network Software Environment, skip to the *Network Software Environment: Installation and Administration* manual.

If you come across a term that you don't understand, try looking in Appendix A, *Glossary*.

## Software Development Organization

Within a software development organization, top-level managers use the NSE for overall product planning. These managers use the NSE browsing features to examine the hierarchy of products and associated objects, such as functional specifications and documentation. By accessing information about the full range of



products developed within their organization, these managers can compose and implement effective software development plans.

## **Release Management**

Release managers, or release engineers, use the NSE to maintain a source hierarchy and to create release environments in preparation for the release of a software product. The NSE provides a platform from which release managers can better organize and integrate hierarchies of files and other objects associated with the development and release of software products, even when product development occurs within multi-architecture networks. In addition, the NSE provides a mechanism to retrieve previous revisions of these hierarchies.

If you are a release manager, see the *Network Software Environment: Reference Manual* (this book), and the *Network Software Environment: Installation and Administration* manuals.

## **Software Development Management**

Software development managers, or project leaders, plan and track software development using the NSE. Giving programmers individual software environments aids these managers in scheduling the completion of specific portions of the project. Many developers use integration tools to merge their work into a common environment as development proceeds.

If you are a software development manager, see the *Network Software Environment: Reference Manual* (this book).

## **Program Development**

Software engineers, or programmers, work within their own individual environments. These environments are part of a larger hierarchy of source files, executables, documentation, and other objects. Each programmer's environment may contain a hierarchy of source files and related objects. Programmers develop code and integrate it into the larger hierarchy, permitting access for other programmers and for managers. In addition, NSE variants ease programming for multiple architectures, and a version control system reduces the possibility of lost work and unrecoverable programming errors.

If you are a software engineer, see the *Network Software Environment: Reference Manual* (this book).

## **Documentation Writing**

Documentation writers work within the NSE so that their documents are accessible through the same hierarchy created for

the rest of the software products. Documentation writers may consult functional specifications and source code when appropriate, just as programmers and managers have easy access to the documentation associated with their products.

If you are a documentation writer, see the *Network Software Environment: Reference Manual* (this book).

### Other Development

The NSE provides a platform for a variety of computer-assisted software engineering (CASE) products. These products make use of the NSE by integrating new objects, appropriate to the application, with the NSE. For example, a third-party software product could handle schematics useful for source code or documentation development.

### Customer Support

Customer support personnel may have to provide software support for a number of different versions of many products running on multiple architectures. In this case, customer support personnel benefit from the multi-architecture variant and release-retrieval revision mechanisms available with the NSE, so they can provide timely and accurate consultation with customers and development staff.

If you work in customer support, see the *Network Software Environment: Reference Manual* (this book).

### System Administration

System administrators share overall maintenance of object hierarchies with release engineers and other developers. The NSE's logical organization of the development process prevents some errors, and permits easier correction of errors that do occur in system administration. Administrators consult with developers to bootstrap the NSE, or create its initial object hierarchy.

For more information on administration of the NSE, see the *Network Software Environment: Installation and Administration* manual.

### Typographical Conventions

This book follows these typographical conventions:

- Times Roman font indicates regular text.
- *Times Roman font, Italic face* indicates:
  - 1) Emphasis.
  - 2) The title of a book, chapter, or section.



- 3) A variable name.
- 4) A command argument.
- **Times Roman font, Bold face** indicates:
  - 1) A chapter or section heading.
  - 2) A term listed and defined within the glossary.
- **Courier font** indicates:
  - 1) A listing.
  - 2) A command name.
  - 3) A program name.
  - 4) Text the machine displays, as in a tutorial session.
- **Courier font, Bold face** indicates what the user types during a tutorial session.

## Documentation Set

These are the books in the NSE documentation set:

- *Network Software Environment: Reference Manual*  
(this book)
- *Network Software Environment: Installation and Administration*

# The Browser

|                                          |    |
|------------------------------------------|----|
| The Browser .....                        | 1  |
| 1.1 Starting Up the Browser.....         | 2  |
| 1.2 Browser Window Header.....           | 2  |
| 1.3 Browser Subwindows .....             | 4  |
| Legend Subwindow .....                   | 4  |
| Display Subwindow .....                  | 5  |
| Log Subwindow .....                      | 6  |
| 1.4 Browser Symbols .....                | 7  |
| Environment Symbol .....                 | 8  |
| Component Symbol .....                   | 9  |
| make Target Symbol.....                  | 10 |
| File Symbol.....                         | 11 |
| Link Database Symbol.....                | 12 |
| Variant Symbol .....                     | 13 |
| Symbol Summary .....                     | 14 |
| 1.5 Browser Menus .....                  | 15 |
| Selection Menus.....                     | 17 |
| Window Menus.....                        | 35 |
| Frame Menu .....                         | 40 |
| 1.6 Dialog Boxes and Popup Windows ..... | 42 |
| Dialog Boxes.....                        | 42 |





---

|                                             |    |
|---------------------------------------------|----|
| Confirmation Windows.....                   | 46 |
| History Windows .....                       | 47 |
| Output Windows .....                        | 48 |
| 1.7 Browser Operations and Menu Items ..... | 49 |
| Acquire.....                                | 50 |
| Add Member... ..                            | 53 |
| All Components .....                        | 55 |
| Append.....                                 | 56 |
| Checkin... ..                               | 58 |
| Checkout .....                              | 60 |
| Child Envs.....                             | 62 |
| Clean Up .....                              | 63 |
| Contents .....                              | 64 |
| Dbxtool .....                               | 65 |
| Delete =>.....                              | 67 |
| Descriptors .....                           | 68 |
| Diffs =>.....                               | 69 |
| Dismiss.....                                | 71 |
| (Editor).....                               | 72 |
| Emacs.....                                  | 73 |
| Get Version... ..                           | 75 |
| History .....                               | 76 |
| Kill .....                                  | 78 |
| Latest.....                                 | 79 |
| List =>.....                                | 80 |
| List History .....                          | 89 |
| List My Envs.....                           | 90 |
| List Other's Envs... ..                     | 91 |
| List Public Envs .....                      | 92 |

---

|                                                |     |
|------------------------------------------------|-----|
| List Source Files .....                        | 93  |
| Lock .....                                     | 94  |
| Make => .....                                  | 95  |
| Notifications.....                             | 97  |
| Other Target... ..                             | 98  |
| Parent Envs .....                              | 99  |
| Preserve.....                                  | 100 |
| Previous Queries =>.....                       | 101 |
| Reconcile => .....                             | 102 |
| Recover => .....                               | 103 |
| Register Notification.....                     | 104 |
| Remove .....                                   | 106 |
| Rename... ..                                   | 107 |
| Resolve => .....                               | 111 |
| Resync => .....                                | 112 |
| Root Component .....                           | 113 |
| Shelltool .....                                | 114 |
| Show Output .....                              | 115 |
| Textedit .....                                 | 116 |
| Two Versions.....                              | 118 |
| Uncheckout .....                               | 119 |
| Unlock.....                                    | 120 |
| Unregister.....                                | 121 |
| Variants.....                                  | 122 |
| Version.....                                   | 123 |
| Vi .....                                       | 124 |
| You Specify... ..                              | 125 |
| 1.8 Using A Text Editor with the Browser ..... | 126 |





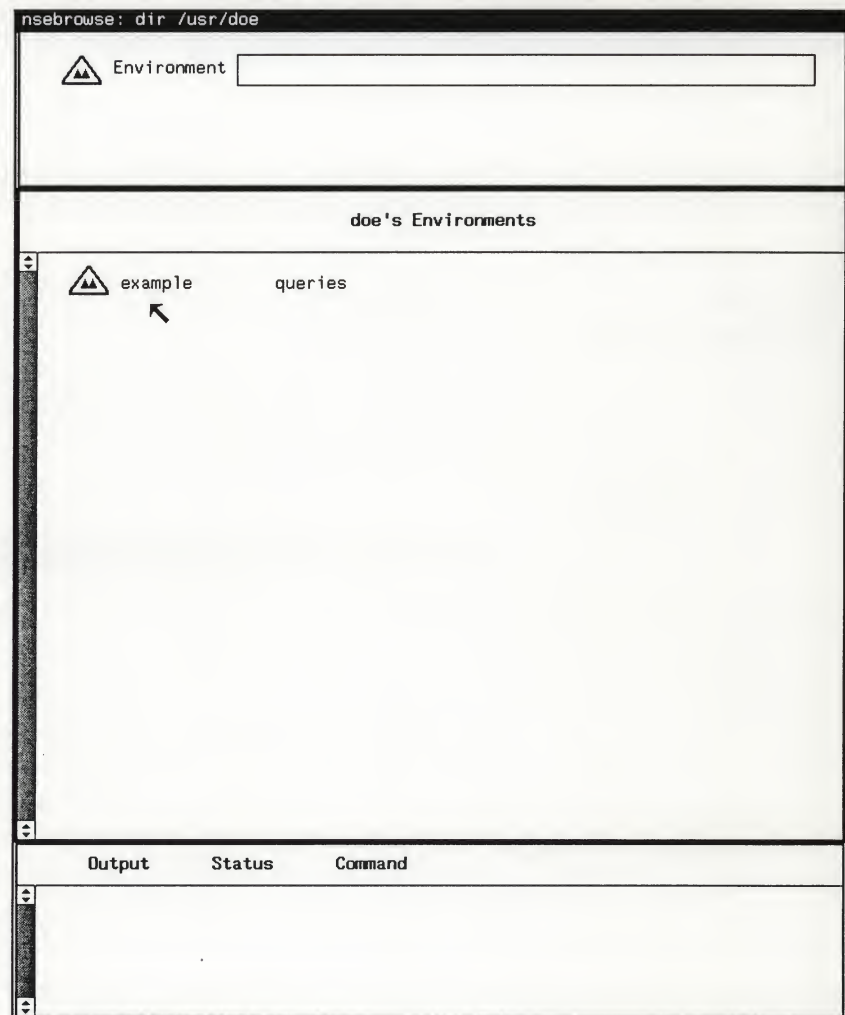
## The Browser

The NSE window- and mouse-based Browser provides a variety of operations for the examination and manipulation of the software development object hierarchy.

Figure 1-1 *The Network Software Environment Browser*



Browser Icon





This chapter describes:

- How to Start Up the Browser
- Browser Subwindows
- Browser Symbols
- Browser Menus
- Dialog Boxes and Confirmation Boxes
- Browser Operations

The NSE Browser runs only on bit-mapped screens. For information on the command-line interface to the NSE, which runs on terminals that don't support a window-based interface, see Chapter 4, *Command-Line Interface*.

## 1.1 Starting Up the Browser

To start up the Browser, type `nsebrowse`, followed by the ampersand symbol `&`, to a shell command prompt as shown in the next figure:

Figure 1-2 *Browser: Start Up*

```
tutorial% nsebrowse &  
tutorial%
```

After a moment, the NSE Browser appears in a window of its own on the screen as shown in the first figure of this chapter.

## 1.2 Browser Window Header

The Browser window header is the stripe that runs across the very top of the Browser window. When the user starts up the Browser, the window header contains the name of the Browser program `nsebrowse`, followed by the name of the directory in which the user started up the Browser.

Figure 1-3 *Browser: Window Header at Start Up*



During the execution of Browser operations, the Browser window header provides status information, reporting the activation of an environment, for example. When activated within an environment, the window header lists the name of the environment, including the variant associated with the environment, in parentheses. The Browser lists the current directory except when the user browses

or operates on a make target. Then, the window header lists the directory name where the target resides on the file system.



### 1.3 Browser Subwindows

The NSE Browser window is divided into three subwindows:

- The Legend
- The Display
- The Log

Some of the subwindows have scroll bars. For more information on how to use scroll bars and other aspects of the SunView window system, see the *Windows and Window-Based Tools: Beginner's Guide*.

#### Legend Subwindow

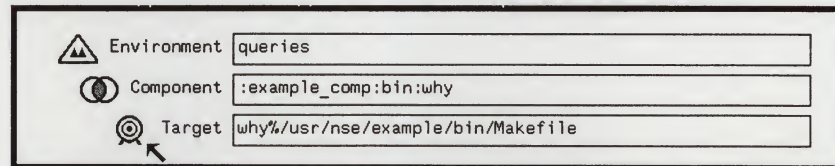
The Legend subwindow is the top subwindow in the Browser. The Legend subwindow shows the current location within the environment and component hierarchies.

Figure 1-4 *Browser: Legend Subwindow*



When the user browses an environment, the Browser lists the name of that environment in the top box of the Legend subwindow. When the user browses a component, the Browser adds another box containing the name of the current component. Similarly, when the user browses a make target, the Browser lists the name of that target in a third box within the Legend.

Figure 1-5 *Browser: Legend Subwindow – Location in Hierarchy*

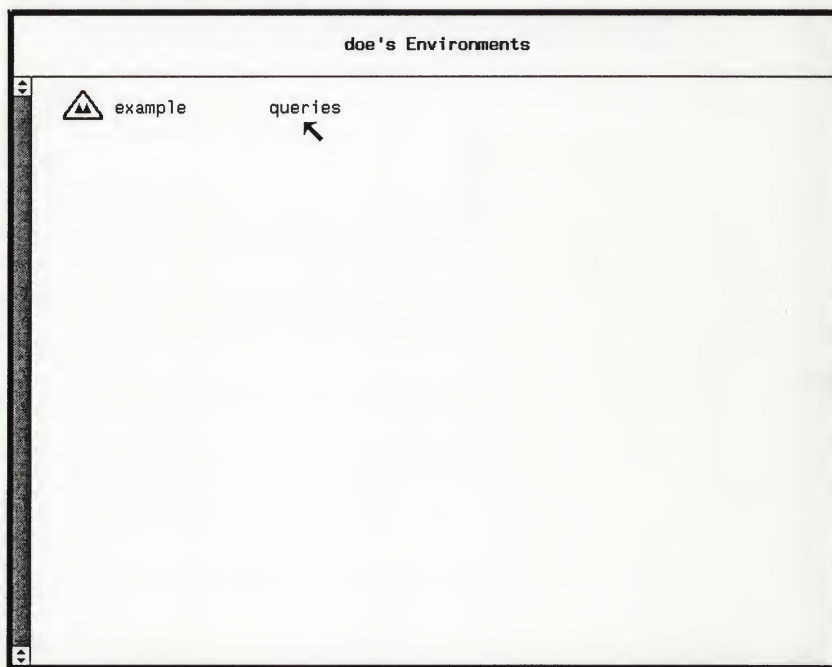


For information on the selection, window, and frame menus accessible from the Legend, see Section 1.5, *Browser Menus*.

## Display Subwindow

The Display subwindow is the middle subwindow of the Browser. The Display subwindow has two parts. The upper portion of the Display shows the title of the query currently displayed in the lower portion of the Display. The lower portion of the Display shows lists of objects, environments, and other aspects of the environment and component hierarchy.

Figure 1-6 *Browser: Display Subwindow*



For example, when the user browses an environment list, that list of environments appears in the Display. Also, when the user browses an environment, component or target, the Display lists its contents. Besides listing the objects contained in environments, components, and targets, the Display can list other things, like variant descriptors and notification requests.

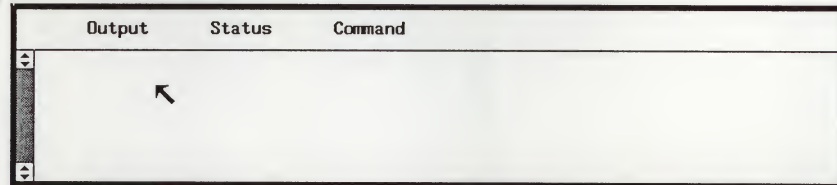
For information on the selection, window, and frame menus accessible from the Display, see Section 1.5, *Browser Menus*.



## Log Subwindow

The Log subwindow is the bottom subwindow of the Browser. The Log subwindow has two parts. The upper portion of the Log is a set of headings for the contents of the lower portion of the Log. The lower portion of the Log is a subwindow that contains a listing of commands executed by the Browser, as well as the state of those commands. The Log Item selection menu provides a way to manipulate these commands, including a way to display their output.

Figure 1-7 *Browser: Log Subwindow*



Items in the Log represent commands executed by the Browser operations the user chooses.

The following values can appear underneath the **Output** heading:

- **New** User hasn't viewed the command output yet.
- **Old** User has viewed existing command output.
- **Viewing** User is viewing the command output.

The following values can appear underneath the **Status** heading:

- **Running** Command is executing.
- **Done** Command has finished and exited with a status of 0, usually signifying successful completion.
- **Failed** Command has terminated because it failed.

A command can fail because the command exits with a non-zero status or because the command died with a certain signal. Using the Log Item menu on a selected Log item, the user can view the output to see exactly why the command failed.

For information on the selection, window, and frame menus accessible from the Log, see Section 1.5, *Browser Menus*.

## 1.4 Browser Symbols

These are the standard symbols used in the Browser:

- Environment
- Component
- make Target
- File
- Link Database
- Variant

Browser symbols appear in the Legend and Display, but not in the Log.



## Environment Symbol

The environment symbol always appears in the Browser Legend to the left of the word `Environment`. The environment symbol is a graphical representation of the individual file system provided by an environment. This figure shows the environment symbol in the left margin and its appearance in the Legend subwindow on the right:



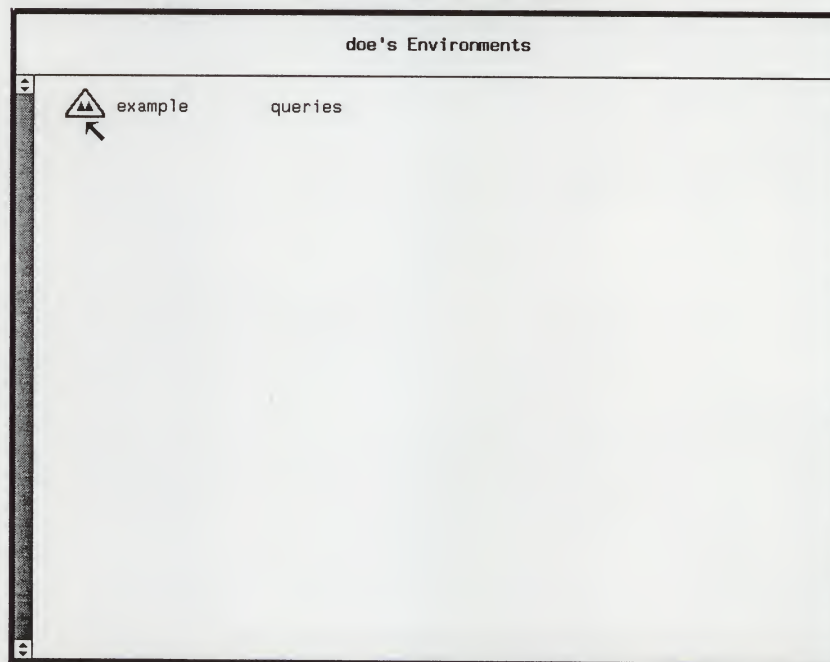
Environment  
Symbol

Figure 1-8 *Browser: Environment Symbol and Legend*



The Browser environment symbol may appear in the Display as well when the Display shows a list of environments:

Figure 1-9 *Browser: Environment Symbol in Display*



## Component Symbol

The component symbol appears in the Legend to the left of the word **Component**, after a user lists the components of an environment. The name of the environment containing the listed components appears in the box to the right of the word **Environment**. The environment symbol is a graphical representation of set membership because components are sets of development objects. This figure shows the component symbol in the left margin and its appearance in the Legend subwindow on the right:

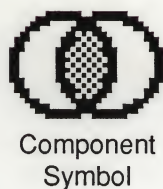
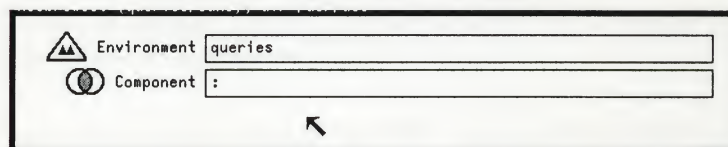
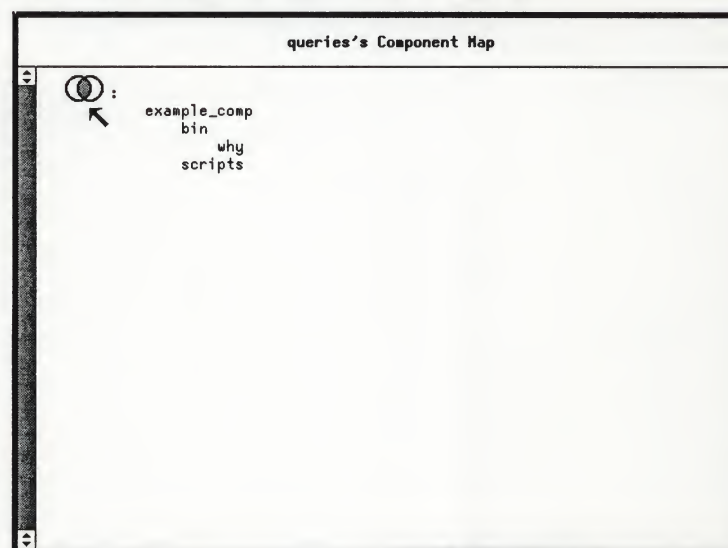


Figure 1-10 *Browser: Component Symbol and Legend*



The Browser component symbol may appear in the Display as well to show the component hierarchy:

Figure 1-11 *Browser: Component Symbol in Display*

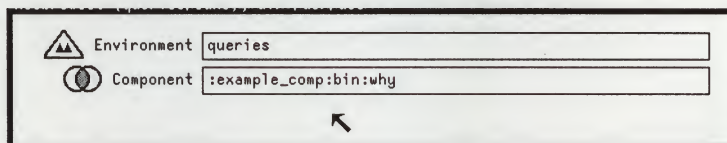




## make Target Symbol

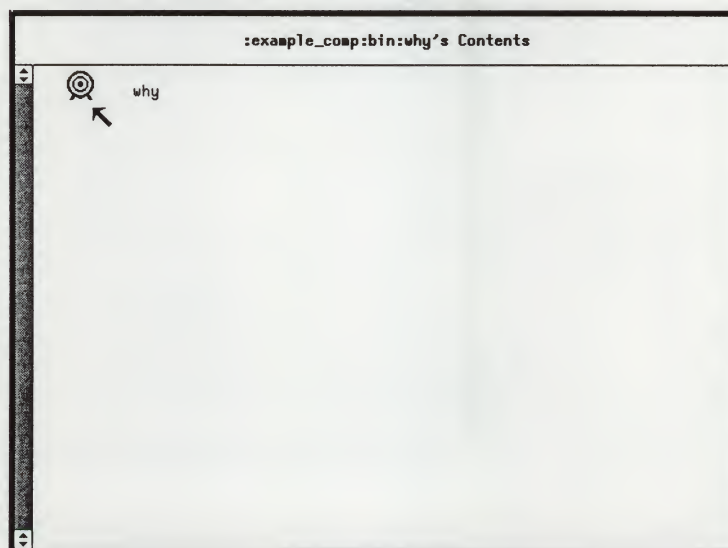
The Browser make target symbol appears in the Legend when the Browser lists the source files of a make target. The name of the target appears in the box to the right of the target symbol, and the name of the environment and component that contain the target appear in boxes to the right of their respective symbols. The target symbol is a graphical representation of a target like that used for shooting arrows. This figure shows the make target symbol in the left margin and its appearance in the Legend subwindow on the right:

Figure 1-12 *Browser: make Target Symbol and Legend*



The make target symbol appears in the Display when a target appears in the list of the contents of a component.

Figure 1-13 *Browser: make Target Symbol in Display*



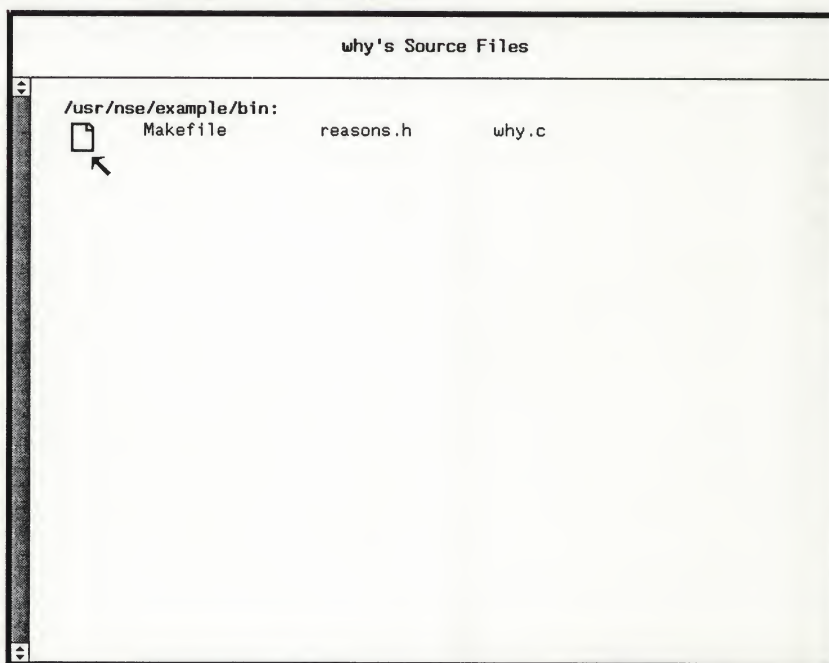
## File Symbol

The Browser file symbol never appears in the Legend. However, the file symbol appears in the Display when listing the contents of a component, or listing the source files of a target. The file symbol is a graphical representation of the kind of file stored in a file cabinet. This figure shows the file symbol in the left margin and its appearance in the Display subwindow on the right:

Figure 1-14 *Browser: File Symbol in Display*



File  
Symbol





## Link Database Symbol

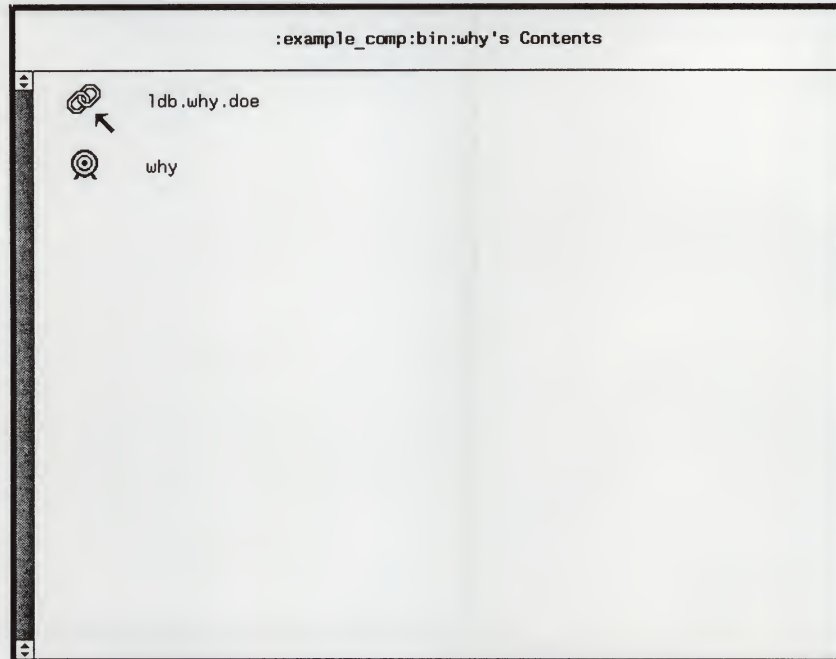
The Browser link database symbol never appears in the Legend. However, the link database symbol appears in the Display when a link database is in the list of the contents of a component. The link database symbol is a graphical representation of the “links” in a section of chain. This figure shows the link database symbol in the left margin and its appearance in the Display subwindow on the right:



Link  
Database  
Symbol

Figure 1-15

*Browser: Link Database Symbol in Display*



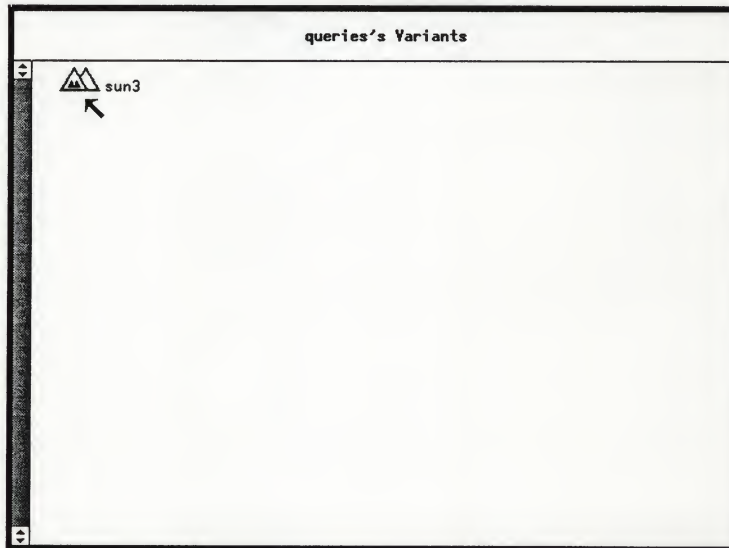
## Variant Symbol

The Browser variant symbol never appears in the Legend. However, the variant symbol appears in the Display when the user lists the variants associated with an environment. The variant symbol is a graphical representation of the way that variants form layers within an environment. This figure shows the variant symbol in the left margin and its appearance in the Display subwindow on the right:



Variant  
Symbol

Figure 1-16 *Browser: Variant Symbol in Display*











## Symbol Summary

In summary, here is a table of the standard Browser symbols:

Table 1-1 *Browser Symbols*

| <i>Description</i>   | <i>Symbol</i>                                                                       | <i>Command-Line Name</i> |
|----------------------|-------------------------------------------------------------------------------------|--------------------------|
| <i>Environment</i>   |    | <i>None</i>              |
| <i>Component</i>     |    | Comp                     |
| make <i>Target</i>   |    | Targ                     |
| <i>File</i>          |    | File                     |
| <i>Link Database</i> |  | Linkdb                   |
| <i>Variant</i>       |  | <i>None</i>              |

The command-line name is the word that the NSE uses to represent the object. Within the command-line interface, and sometimes within the Browser, the user types this name on the keyboard to represent the desired object type. Because environments and variants are not “objects” in the sense of components, targets, files, and link databases, they don’t require a command-line type name.

## 1.5 Browser Menus

These are the types of Browser menus:

- Selection menus
- Window menus
- Frame menu

When a menu item contains an ellipsis (. . .), choosing that item causes a popup window to appear so that the user can specify additional information necessary to execute the operation. For more information on popup windows, see Section 1.6, *Dialog Boxes and Popup Windows*.

When a menu item ends in a pull-right symbol that looks something like this =>, dragging the cursor through the pull-right symbol causes a pull-right menu to appear. Then, the user can select one of the items on the pull-right menu. The specific operation of these pull-right menus depends on the way the user sets up SunView window system defaults. For more information, see the *Windows and Window-Based Tools: Beginner's Guide*.

The Browser permits the user to execute operations one after another, "in parallel," without waiting for each operation to complete execution. This is an advantage because the user can continue working without waiting for each operation to finish. However, to a user unaware of this parallel execution capability, the behavior resulting from execution of several successive Browser operations could occasionally produce results unintended by the user.

Browser menus permit the user to specify two kinds of operations:

- Query operations
- Command operations

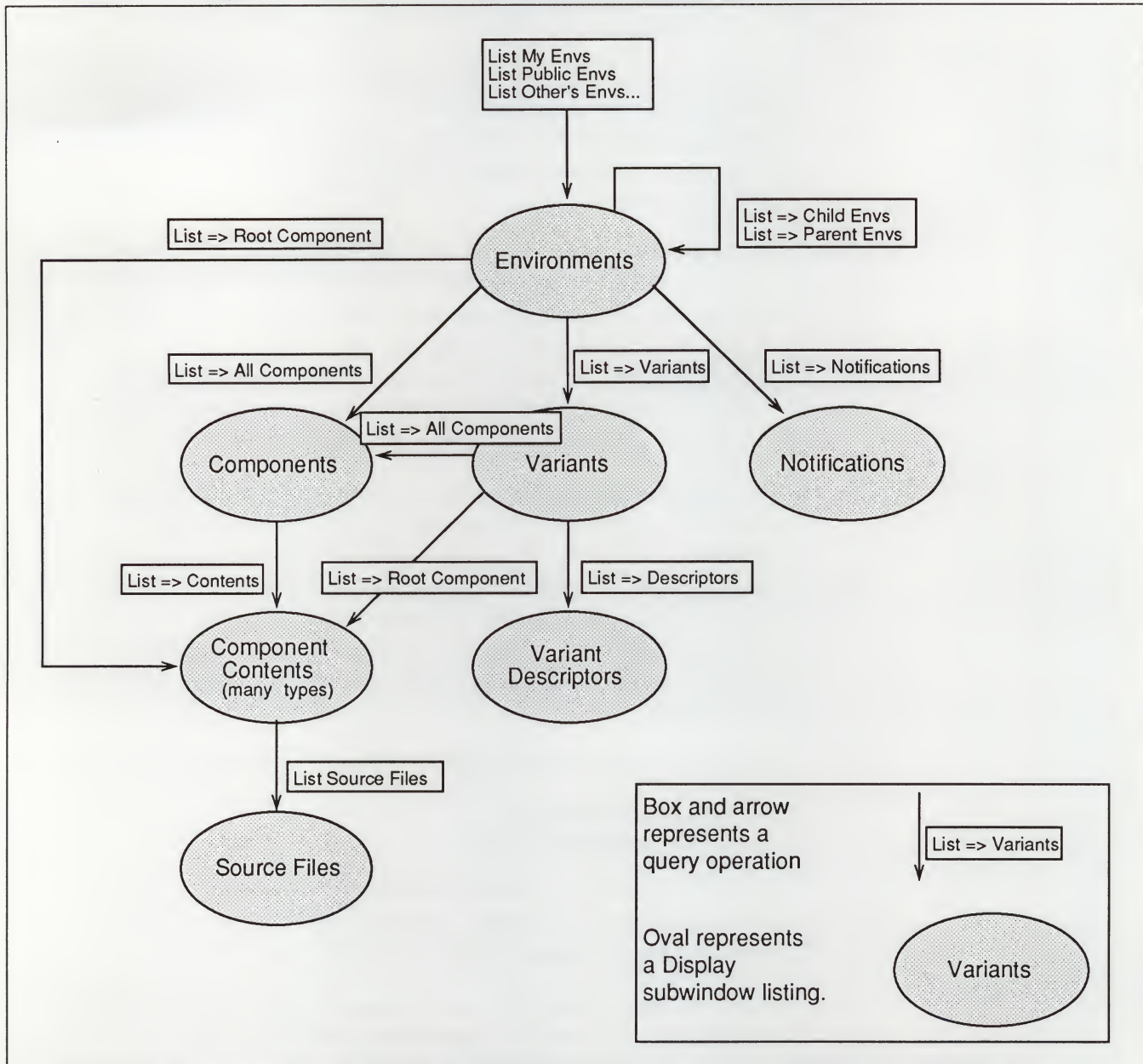
All of the menu items that begin with the word `List` represent query operations, or queries. Except for the `List History` item, all of these queries show their results in the Display.

All of the menu items that aren't query operations are command operations. All command operations generate entries in the Log.



When the user navigates through the query operations of the Browser, the following figure may prove useful.

Figure 1-17 *Browser: Queries and How They Interrelate*



## Selection Menus

Selection menus appear when the user holds down the right mouse button after making a selection in one of the Browser subwindows. The user can make a selection and bring up a Selection menu in any of the subwindows.

The list of Selection menus includes these menus, including one menu specific to each of the object types available in the Browser:

- Environment menu
- Component menu
- Target menu
- File menu
- Link Database menu
- Variant menu
- Notification menu
- Log Item menu

**Note:** The user may switch the SunView defaults for the mouse so that the standard left mouse button may act like the right mouse button, and *vice versa*. For the purpose of this manual, please assume references are to the standard SunView mouse defaults.

The user can select Browser text by clicking the left mouse button on some portion of the text. To “extend” the selection so the selection includes more text, the user can click on the additional text with the middle mouse button. This allows the user to specify that the Browser operate on more than one object at a time, for example, to check out more than one file from the Version Control System. Some operations apply to one object only, so when the user applies one of these operations to more than one object, the operation produces a warning.

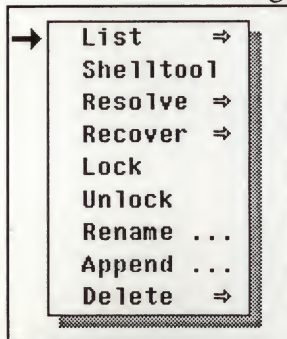
The user can remove a selection from a group of selected objects by clicking on it with the middle mouse button. The user can remove selections from all of the selected objects by clicking the left mouse button over a blank area of the screen, in effect, a nil selection.



## Environment Menu

The Environment menu operates on environments. To access the Environment menu, the user selects an environment name in the Legend or Display subwindow and holds down the right mouse button.

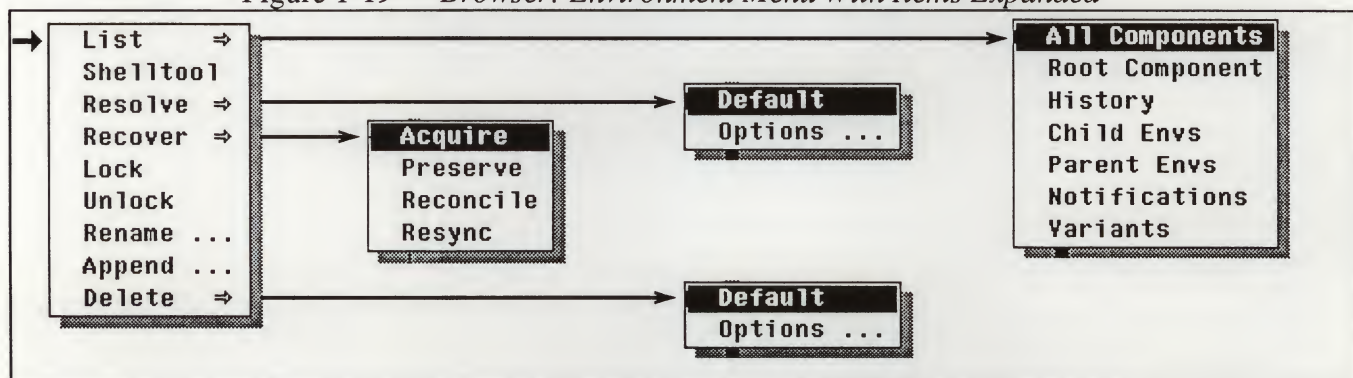
Figure 1-18

*Browser: Environment Menu*

Magnified Image



Certain of the Environment menu items pull right to other menus.

Figure 1-19 *Browser: Environment Menu With Items Expanded*

When the user browses an environment, the Browser automatically activates that environment. The Browser automatically deactivates the environment when the user starts to browse another environment.

For detailed information on the Environment menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Environment menu operation appears below:

List => Pulls right to the following menu items:

- All Components
- Root Component
- History
- Child Envs
- Parent Envs
- Notifications
- Variants

The List => All Components item lists all of the components of the selected environment in the Display.

The List => Root Component item lists only the root component of the selected environment.

The List => History item causes a popup window to appear which lists the history information associated with the selected environment.

The List => Child Envs item lists the names of the child environments of the selected environment.

The List => Parent Envs item lists the names of the parent environments of the selected environment.

The List => Notifications item lists the notification requests associated with the selected environment.

The List => Variants item lists the names of the variants associated with the selected environment.

Shelltool Invokes a shelltool activated within the current environment.

Resolve => Pulls right to the following menu items:

- Default
- Options....

The Resolve => item resolves the changes made in the selected environment with the changes made in its parent environment.

Recover => Pulls right to the following menu items:

- Acquire
- Preserve
- Reconcile



Resync

Append

The `Recover =>` item permits the recovery of a failed command when the command is one of the commands listed as items in the `Recover =>` pull-right menu.

**Lock** Locks the selected environment so that no other user can make changes to the environment until someone releases the lock on that environment.

**Unlock** Releases a lock that the user placed on the selected environment.

**Rename...** Renames the selected environment. The user specifies the new environment name in a dialog box.

**Append...** Appends an object to the selected environment. The user specifies the source environment, type of the object, and object name in a dialog box.

**Delete =>** Pulls right to the following menu items:

Default

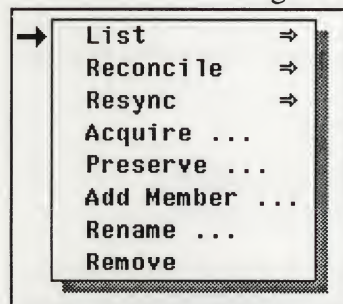
Options...

The `Delete =>` item deletes the selected environment.

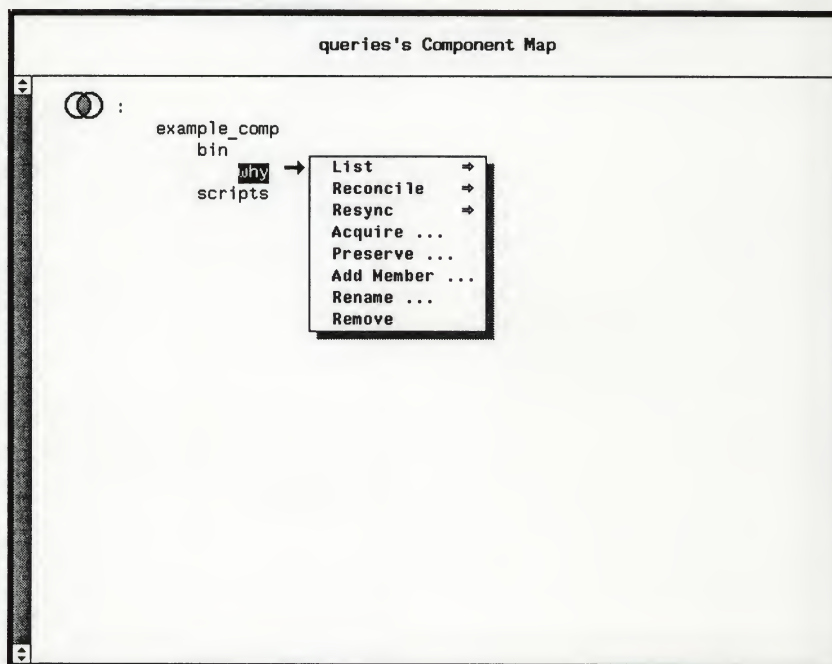
## Component Menu

The Component menu operates on components. To access the Component menu, the user selects a component name in the Legend or Display subwindow and holds down the right mouse button.

Figure 1-20

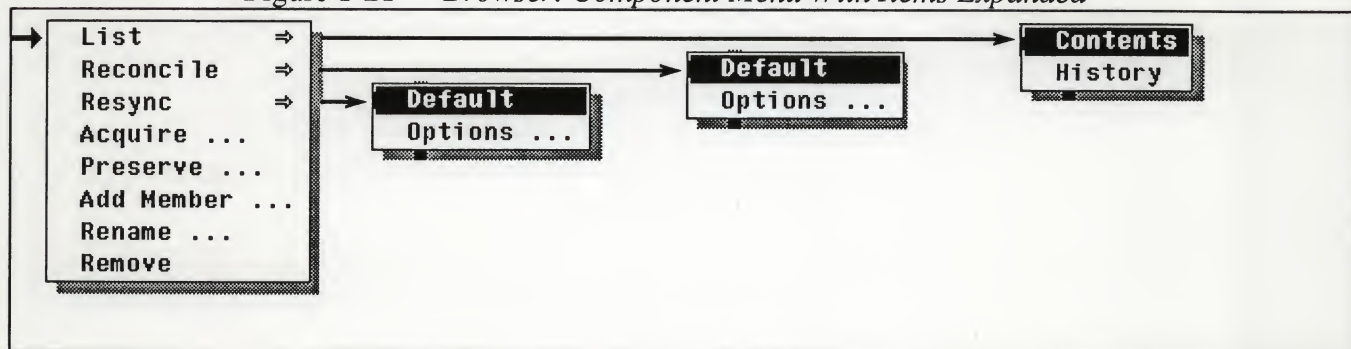
*Browser: Component Menu*

Magnified Image



Certain of the Component menu items pull right to other menus.

Figure 1-21

*Browser: Component Menu With Items Expanded*

For detailed information on the Component menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Component menu operation appears below:

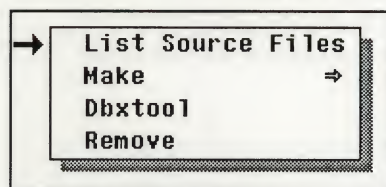


- List => Pulls right to the following menu items:  
Contents  
History  
The List => item either lists the contents, or generates a popup window containing the revision history of the selected component.
- Reconcile => Pulls right to the following menu items:  
Default  
Options...  
The Reconcile => item reconciles the selected component in the currently activated environment to its parent environment.
- Resync => Pulls right to the following menu items:  
Default  
Options...  
The Resync => item resyncs the selected component in the currently activated environment from its parent environment.
- Acquire... Acquires the selected component from the currently activated environment into a child environment. The user specifies the child environment name in a dialog box.
- Preserve... Makes a revision preserving the contents of the selected component in the currently activated environment. The user specifies the name of the revision, some revision comments, and whether or not to preserve only the top-level component in a dialog box.
- Add Member... Adds an object to the selected component. The user specifies the name and type of the object in a dialog box.
- Rename... Renames the selected component. The user specifies the new name for the component in a dialog box.
- Remove Removes the selected component from the currently activated environment.

## Target Menu

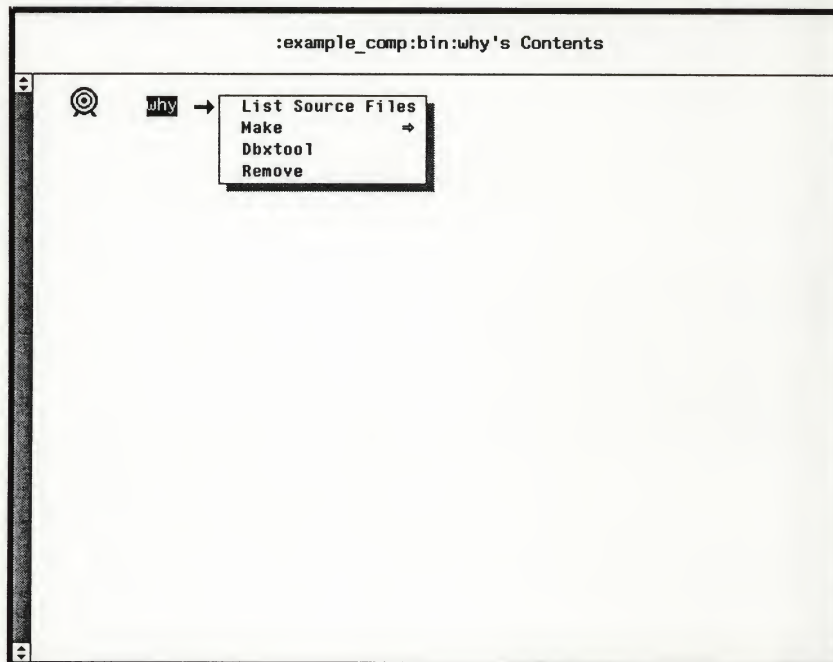
The Target menu operates on make targets. To access the Target menu, the user selects a target name in the Legend or Display subwindow and holds down the right mouse button.

Figure 1-22



Magnified Image

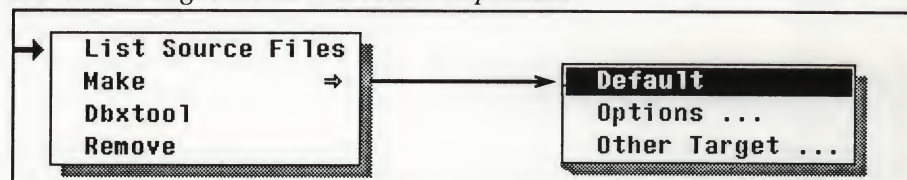
Browser: Target Menu



One of the Target menu items pulls right to another menu.

Figure 1-23

Browser: Target Menu With Item Expanded



For detailed information on the Target menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Target menu operation appears below:

**List Source Files** Lists the names of the source files the selected make target contains.

**Make =>** Pulls right to the following menu items:  
 Default  
 Options...  
 Other Target...



The `Make =>` item builds the selected target using the `make` program.

`Dbxtool` Invokes `dbxtool` on the selected target.

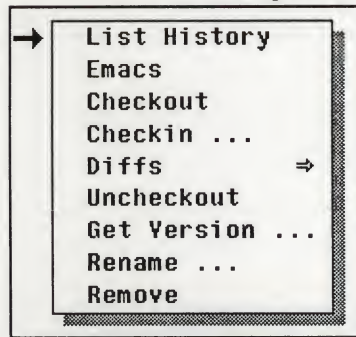
`Remove` Removes the selected target from the current component in the currently activated environment.

## File Menu

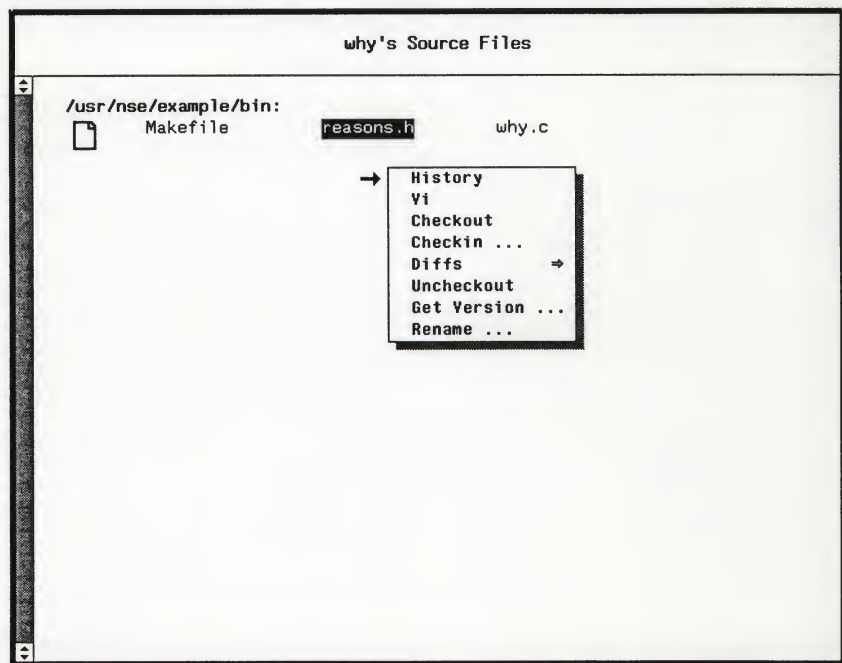
The File menu operates on files. To access the File menu, the user selects a filename in the Legend or Display subwindow and holds down the right mouse button.

The items in this menu provide a window-based interface to the NSE's Version Control System.

Figure 1-24

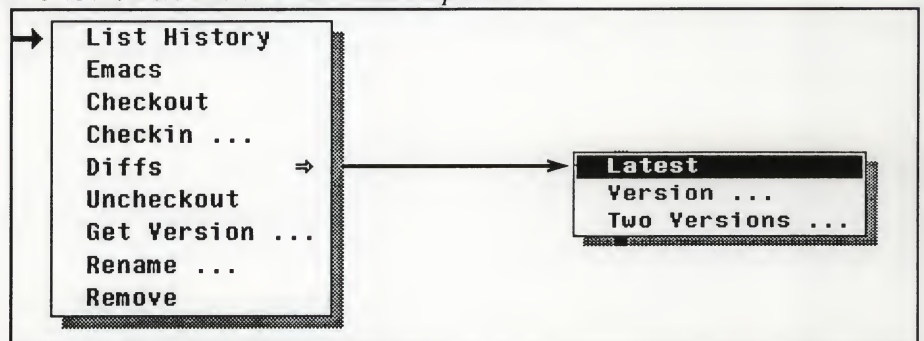
*Browser: File Menu*

Magnified Image



One of the File menu items pulls right to another menu.

Figure 1-25

*Browser: File Menu With Item Expanded*

For detailed information on the File menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each File menu operation appears below:



List History Causes a popup window to appear which lists the Version Control history for the selected file.

(Editor) The default menu item is Vi, but the Browser also supports Emacs and Textedit menu items. In any case, this editor menu item causes a popup window to appear that executes the editor on the selected file. For information on how to change the editor that this item invokes, see Section 1.8, *Using a Text Editor with the Browser*.

Checkout Checks out the selected file from the Version Control System, so that the user may edit the file.

Checkin... Checks in the selected file to the Version Control System, storing the changes the developer made to that file. The user specifies a version comment in a dialog box. Checking in the selected file makes the changes made to the file visible in all of the variants associated with the current environment.

Diffs => Pulls right to the following items:

Latest

Version...

Two Versions...

The Diffs => Latest item compares the current working version with the latest version of the file.

The Diffs => Version... item compares the version of the selected file specified in a dialog box with the latest version of the file.

The Diffs => Two Versions... item compares the two versions of the selected file that the user specifies in a dialog box.

Uncheckout Discards the current working version of the selected file without checking that file in, so the developer should check the file out again when continuing work on that file.

Get Version... Obtains a named version of the selected file by specifying the version name in a dialog box.

Rename... Renames the selected file to a new filename that the user specifies in a dialog box. The Rename... item enables the Version Control System to track the history of the selected file so that users can later

recreate past versions of the file under the old name of that file.

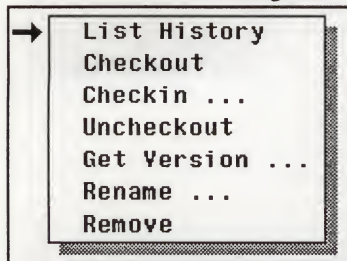


## Link Database Menu

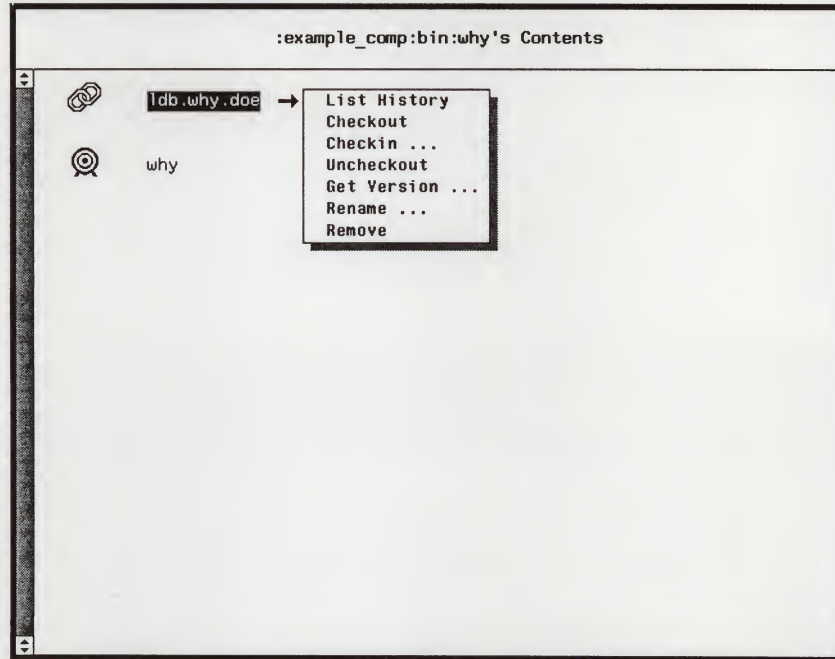
The Link Database menu operates on link databases. To access the Link Database menu, the user selects a link database name in the Display subwindow and holds down the right mouse button.

Figure 1-26

*Browser: Link Database Menu*



Magnified Image



For detailed information on the Link Database menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Link Database menu operation appears below:

- |              |                                                                                                                                                                                                                                                                                                                                      |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| List History | Lists the history information associated with the selected database in a popup window.                                                                                                                                                                                                                                               |
| Checkout     | Checks out the selected link database from the Version Control System, so that the user may access or change the link database.                                                                                                                                                                                                      |
| Checkin...   | Checks in the selected link database to the Version Control System, storing the changes the developer made to that database. Checking in the selected link database makes the changes made to the file visible in all of the variants associated with the current environment. The user specifies a version comment in a dialog box. |
| Uncheckout   | Discards the current working version of the selected link database without checking that link                                                                                                                                                                                                                                        |

database in, so the developer should check the link database out again when continuing work on that link database.

**Get Version...**                      Obtains a named version of the selected link database by specifying the version name in a dialog box.

**Rename...**                              Renames the selected link database to a new link database name that the user specifies in a dialog box. The **Rename...** item enables the Version Control System to track the history of the selected link database so that users can later recreate past versions of the link database under the old name of that link database.

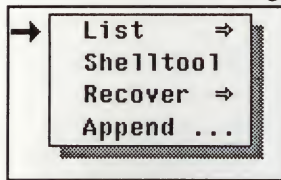
**Remove**                                      Removes the selected link database from the current component in the currently activated environment.



## Variant Menu

The Variant menu operates on variants of environments. To access the Variant menu, the user selects a variant name in the Display subwindow and holds down the right mouse button.

Figure 1-27

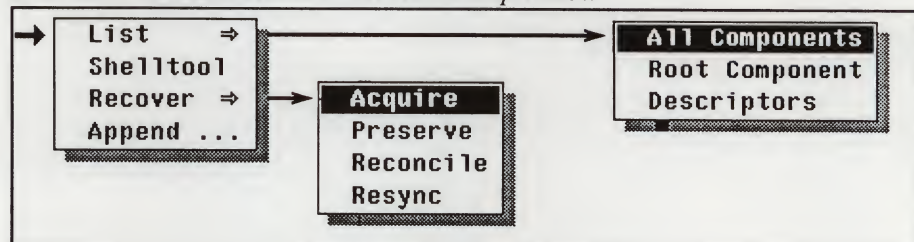
*Browser: Variant Menu*

Magnified Image



Certain of the Variant menu items pull right to other menus.

Figure 1-28

*Browser: Variant Menu With Items Expanded*

For detailed information on the Variant menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Variant menu operation appears below:

|                |                                          |
|----------------|------------------------------------------|
| List =>        | Pulls right to the following menu items: |
| All Components |                                          |
| Root Component |                                          |
| Descriptors    |                                          |

The `List =>` item lists one of the items the user chooses from the pull-right menu.

`Shelltool`                      Invokes a `shelltool` activated within the selected variant associated with the current environment.

`Recover =>`                      Pulls right to the following menu items:

- `Acquire`
- `Preserve`
- `Reconcile`
- `Resync`
- `Append`

The `Recover =>` item permits the recovery of a failed command when the command is one of the commands listed as items in the `Recover =>` pull-right menu.

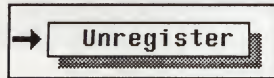
`Append...`                      Appends an object to the selected variant associated with the current environment. The user specifies the source environment, type of the object, and object name in a dialog box.



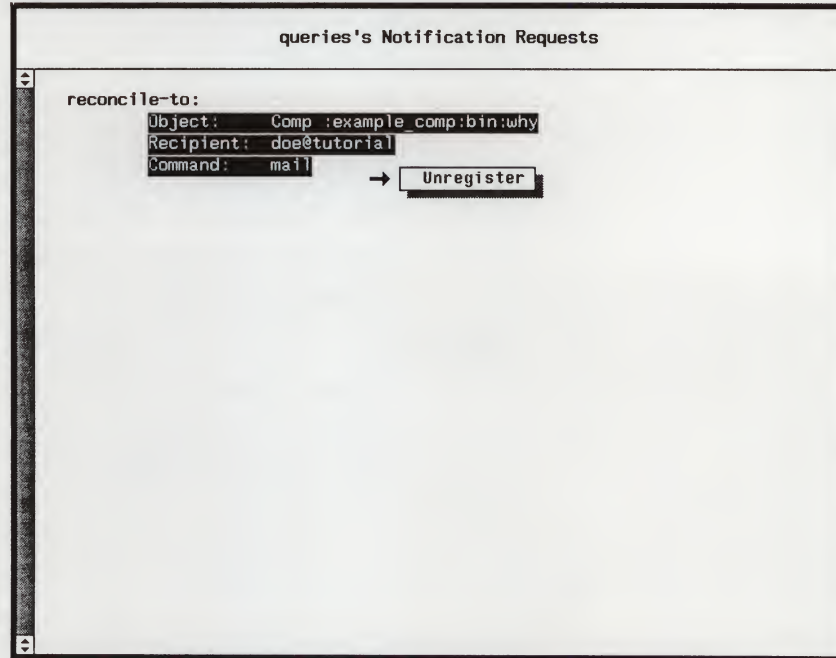
## Notification Menu

The Notification menu operates on notification requests. To access the Notification menu, the user selects the name of a notification request in the Display subwindow and holds down the right mouse button.

Figure 1-29

*Browser: Notification Menu*

Magnified Image



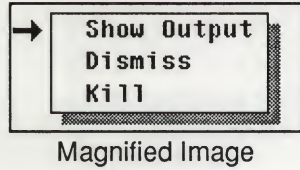
For detailed information on the Notification menu operation, see the description of the operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of the Notification menu operation appears below:

|            |                                   |
|------------|-----------------------------------|
| Unregister | Cancels the selected notification |
| request.   |                                   |

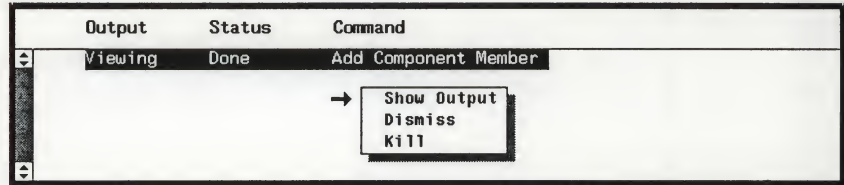
## Log Item Menu

The Log Item menu operates on items in the Log. To access the Log Item menu, the user selects a log item in the Log subwindow and holds down the right mouse button.

Figure 1-30



Browser: Log Item Menu



For detailed information on the Log Item menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Log Item menu operation appears below:

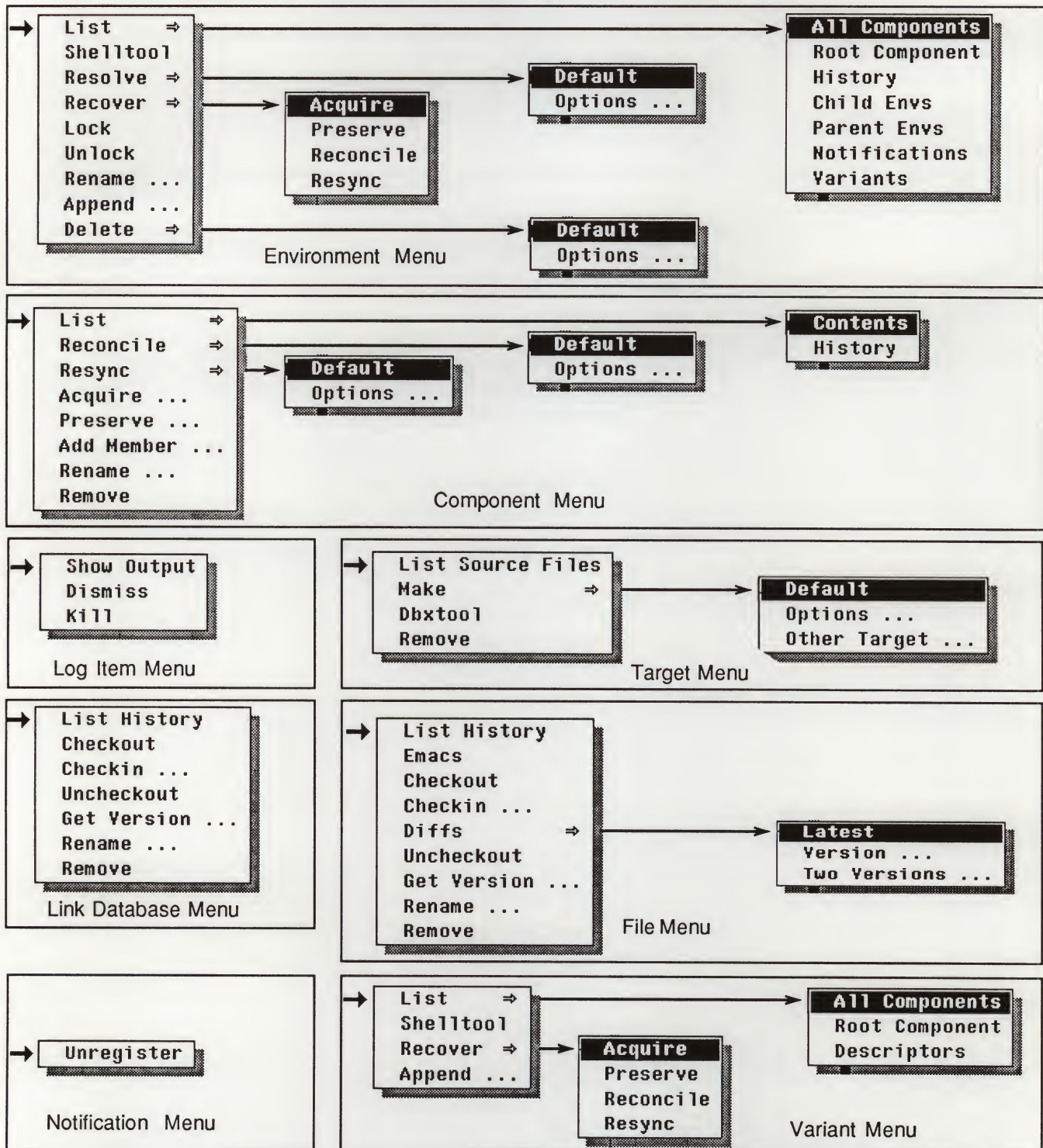
- |             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| Show Output | Causes a popup window to appear which contains the output from the command represented by the selected item in the Log. |
| Dismiss     | Clears the selected log item from the list of log items in the Log.                                                     |
| Kill        | Aborts the execution of the command represented by the selected item in the Log.                                        |



## Summary of Selection Menus

The figure below diagrams the operations available from all of the selection menus.

Figure 1-31 *Browser: Summary of Selection Menus*



## Window Menus

Window menus are the menus that appear when the user holds down the right mouse button in a subwindow in which there are no selections. There are three types of window menus:

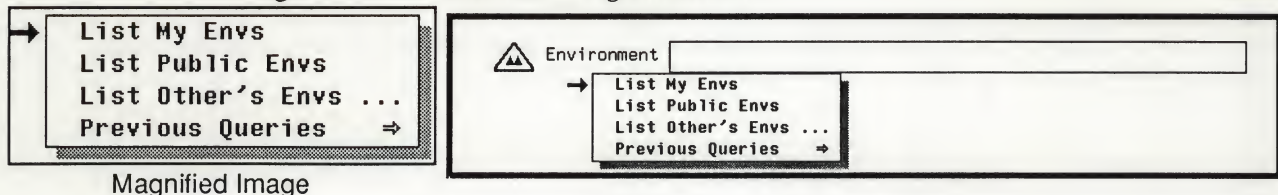
- Legend Menu
- Display Menu
- Log Menu

### Legend Menu

The Legend menu appears when the user holds down the right mouse button in a Legend subwindow that contains no selections.

Figure 1-32

*Browser: Legend Menu*



For more information on the Legend menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Legend menu operation appears below:

- |                      |                                                                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| List My Envs         | Provides a list, in the Display, of the environments owned by the user who invokes the Browser.                                                                                                            |
| List Public Envs     | Provides a list, in the Display, of the public environments, that is, the environments owned by the administrative user Nse.                                                                               |
| List Other's Envs... | Provides a list, in the Display, of the environments owned by another user. The user specifies the username of the other user in a dialog box.                                                             |
| Previous Queries =>  | Pulls right to a list of up to ten of the most recent browsing operations the user invoked in the Browser. By choosing one of the previous queries, the user can return the Browser to a previous display. |



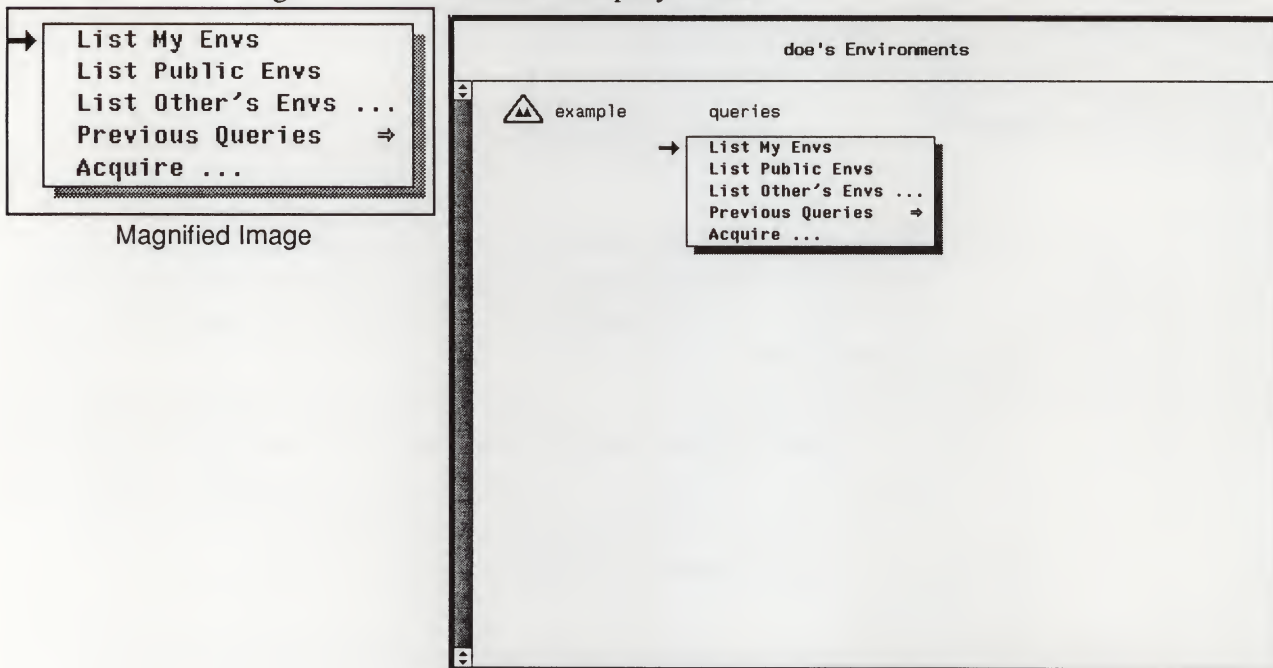
## Display Menu

The Display menu appears when the user holds down the right mouse button in a Display subwindow that contains no selections.

The Display menu is the same as the legend menu except that sometimes the Browser adds a “creation” item to the Display menu. This creation item provides the user with a way to create another object similar to the ones shown in the Display. Depending on context, the Display menu items change so the user can:

- Acquire a component from one environment to another.
- Add a new object to a component.
- Register a notification request.

Figure 1-33 *Browser: Display Menu*



Magnified Image

For more information on the Display menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Display menu operation appears below:

- |                  |                                                                                                                              |
|------------------|------------------------------------------------------------------------------------------------------------------------------|
| List My Envs     | Provides a list, in the Display, of the environments owned by the user who invokes the Browser.                              |
| List Public Envs | Provides a list, in the Display, of the public environments, that is, the environments owned by the administrative user Nse. |

**List Other's Envs...**

Provides a list, in the Display, of the environments owned by another user. The user specifies the username of the other user in a dialog box.

**Previous Queries =>** Pulls right to a list of several of the most recent browsing operations the user invoked in the Browser. By choosing one of the previous queries, the user can return the Browser to a previous display.

**Acquire...** Acquires a component from one environment to another environment. The user specifies the source parent environment, the destination child environment, and the name of the component to acquire in a dialog box.

**Add Component Member...** Adds an object to the current component. The user specifies the name and type of the object in a dialog box.

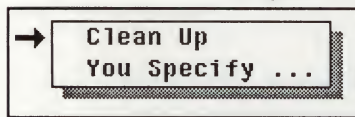
**Register Notification...** Requests notification of the completion of an event. The user specifies the event type, the object type, and the object name for most events in a dialog box.



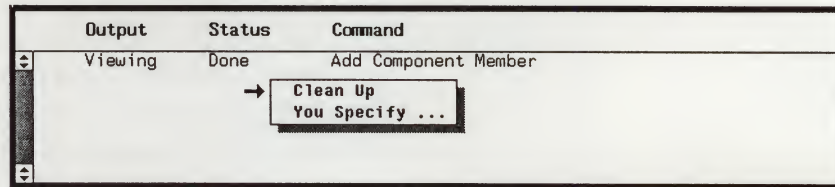
## Log Menu

The Log menu appears when the user holds down the right mouse button in a Log subwindow that contains no selections.

Figure 1-34

*Browser: Log Menu*

Magnified Image



For more information on the Log menu operations, see the description of each operation in Section 1.7, *Browser Operations and Menu Items*. A brief summary of each Log menu operation appears below:

- Clean Up**                      Clears all of the log items from the Log which represent commands that are no longer running.
- You Specify...**            Permits the user to execute a command of the user's choice in a popup window. The user specifies this command in a dialog box.



---

# Network Software Environment<sup>™</sup> Reference Manual

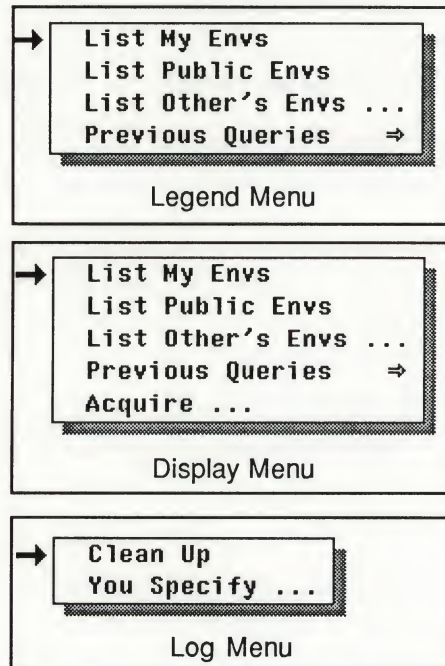




## Summary of Window Menus

The figure below illustrates the operations available from all of the Window menus.

Figure 1-35 *Browser: Summary of Window Menus*



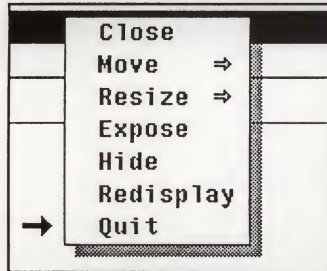


## Frame Menu

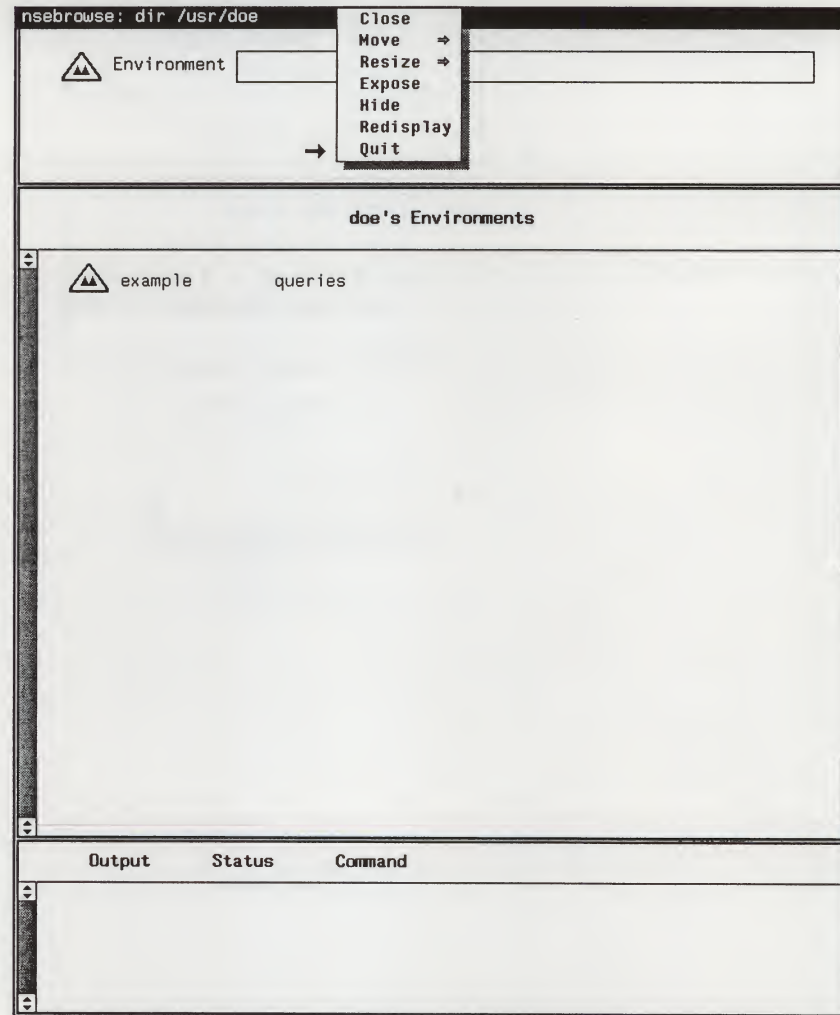
The Frame menu appears when the user holds down the right mouse button in the window frame (including the window header and the border). The Frame menu permits the user to manipulate the window attributes of the Browser.

Figure 1-36

### *Browser: Frame Menu*

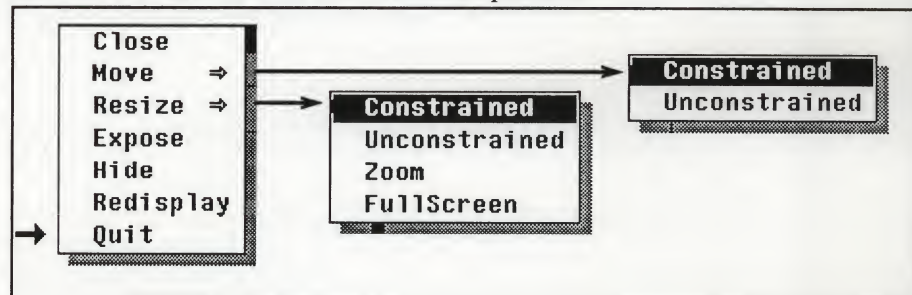


Magnified Image



Certain of the Frame menu items pull right to other menus.

Figure 1-37 *Browser: Frame Menu With Items Expanded*



For more information on the Frame menu and other features of SunView, the Sun window system, see the *Windows and Window-Based Tools: Beginner's Guide*.



## 1.6 Dialog Boxes and Popup Windows

The Browser generates a variety of popup windows when an operation either requires input from the user or generates output for the user. Here is a list of the kinds of popup windows generated by the Browser:

- Dialog Boxes
- Confirmation and Error Windows
- History Windows
- Output Windows

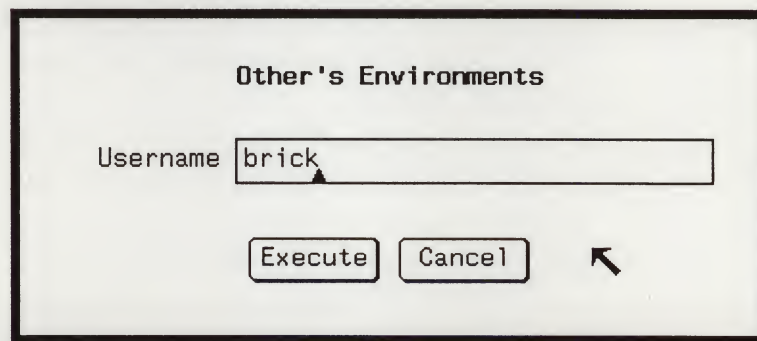
The History and Output windows start as a size of 80 x 34 characters, the standard window size. However, the user can resize these windows, and the Browser remembers the new size and placement for subsequent instances of the same type of window. For dialog boxes, the Browser only remembers the location of the window across invocations of the Browser because the dialog box size varies. Confirmation windows always appear centered on the screen.

### Dialog Boxes

Dialog boxes provide a way for the user to enter arguments to Browser operations, analogous to the options and arguments a user appends to commands when using the command-line interface.

Dialog boxes appear when a user chooses a menu item that ends in ellipses (...). A typical example of a dialog box appears when the user selects the `List Other's Envs...` item of the Legend or Display menu.

Figure 1-38 *Browser: Dialog Box*



To enter the argument, in this case the username `brick`, into the text entry box to the right of the label `Username`, move the cursor within the frame of the dialog box and type the username.

Dialog boxes execute the desired operation when the user clicks the left mouse button on the `Execute` graphical button. The user can cancel an operation while within a dialog box by clicking on the `Cancel` graphical button.

Some Browser operations permit the user to specify more than one argument or option. These operations require a more versatile, and possibly multiple-window dialog box. A typical example of a multiple-window dialog box appears when the user selects the `Acquire...` item of the `Display` menu, which is accessible when listing environments in the `Display`.

Figure 1-39 *Browser: Dialog Box, Parameters Specification*

**Acquire**

☒ Parameters ☐ Options

From Parent Env

To Child Env

Component

Child Env: ☒ Create ☐ Exists

To enter the first argument, in this case the parent environment name `queries`, into the top text entry box to the right of the words `From Parent Env`, move the cursor within the frame of the dialog box and type the username. When the user types a carriage return, the focus of text entry switches to the next text entry box so the user can enter the next argument, and so on. When the user types a carriage return in the bottom text entry box, the focus of text entry switches once again to the top text entry box.

Another way to switch the text entry box of the text entry focus is to click the left mouse button within the desired text entry box. The



caret, represented as a little blinking triangle, always indicates the current input focus.

In the case of the little graphical buttons, such as the buttons to the left of the words *Create* and *Exists*, the user can choose one of the group of graphical buttons by clicking the left mouse button on the desired graphical button.

### Parameters

When the multiple-window dialog box first appears, it displays a parameters specification window. The parameters specification window includes most everything the user needs to deal with to execute the operation. The user can type arguments and specify graphical button choices, if any, as described above.

To switch a dialog box from the parameters specification window to the options specification window, click the left mouse button in the little graphical button to the left of the word *Options*.

Figure 1-40 *Browser: Dialog Box, Options Specification*

The screenshot shows a dialog box titled "Acquire". At the top, there are two checkboxes: "Parameters" (unchecked) and "Options" (checked). Below these, there are three rows of options, each with a checked checkbox and an unchecked checkbox:

- Files Will Be: ☒ Shared ☐ Copied
- Output Level: ☒ Normal ☐ Silent
- Command Action: ☒ Execute ☐ Show Only

Below the options are three text input fields:

- Root Dir: [text input field]
- Variant: [text input field]
- From Revision: [text input field]

At the bottom, there are two buttons: "Execute" and "Cancel". To the right of the "Cancel" button is a small graphical button consisting of a square with a diagonal line from the bottom-left to the top-right.

### Options

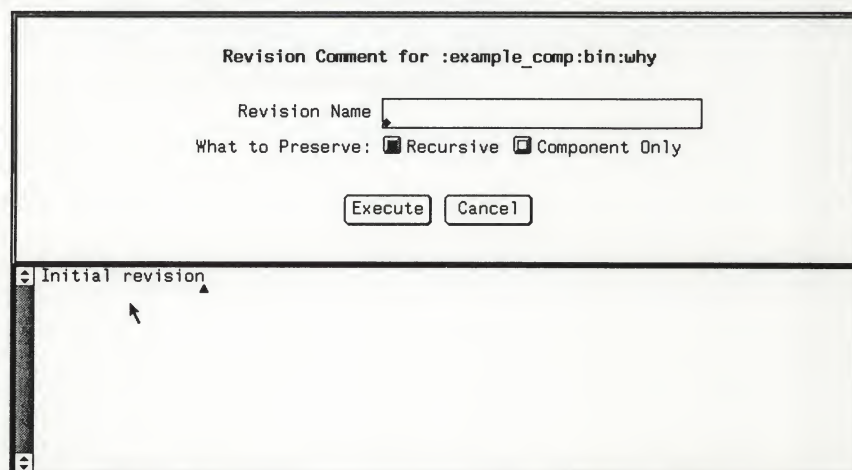
The options specification window is similar to the parameters specification window except that the arguments and graphical button choices are optional. The options specification window of a dialog box permits the user to type optional arguments and specify optional little graphical button choices.

To switch a dialog box from the options specification window to the parameters specification window, click the left mouse button in the little graphical button to the left of the word `Parameters`.

## Comments

In some cases, a dialog box permits the user to specify comments to accompany a Browser operation. For example, the `Preserve . . .` item of the Component menu causes a dialog box to appear. To enter comments into a dialog box that provides a comments subwindow, the user moves the cursor into the comments subwindow, in other words, the lower portion of the dialog box. Then, the user types the comments to associate with the current Browser operation, in the case of this example, the comments associated with the revision of the selected component, produced by the `Preserve . . .` operation.

Figure 1-41 *Browser: Dialog Box, Comments Specification*



The comments subwindow is a text subwindow, so the user can make use of all of the text subwindow editing capabilities. For information on how to edit in a text subwindow or manipulate the window attributes of a dialog box, see the *Windows and Window-Based Tools: Beginner's Guide*.



## Confirmation Windows

Confirmation windows are a degenerate case of dialog boxes in which the user can make a choice about whether to continue or abort the current potentially destructive Browser operation.

For example, the Delete item on the Environment menu causes a confirmation window to appear. Either the user clicks on the left mouse button to confirm the execution of the operation, the deletion of the environment in this example, or the user clicks on the right mouse button to abort the operation, with no environment deletion in the case of this example.

Figure 1-42 *Browser: Confirmation Window*



## History Windows

When a Browser operation generates output about the history of a component, of revisions of a component, or of versions of a file or a link database, the operation causes a popup window to appear that lists the history information associated with the selected object.

For example, the `List History` item on the File menu causes a history window to appear displaying history information about the selected file.

Figure 1-43 *Browser: History Window*



For information on how to manipulate the window attributes of a history window, see the *Windows and Window-Based Tools: Beginner's Guide*.

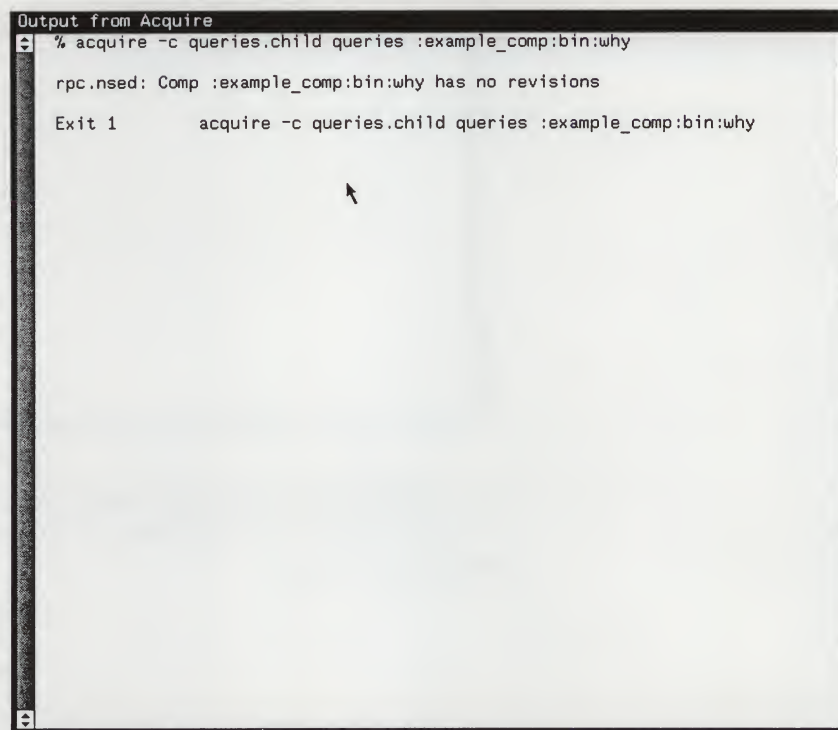


## Output Windows

When the Browser executes a command, it makes an entry in the Log window and captures the output of the command for the user to view when convenient. The user selects an item in the Log window and chooses the `Show Output` menu item to view, in an output window, the output associated with the command.

For example, when the user chooses the `Acquire...` item of the `Display` menu, which is accessible when listing environments in the `Display`, the command associated with the `Acquire...` operation might fail to acquire the desired component because the component had not yet been preserved. In this case, the Log subwindow indicates the failure of the command. The user could then select the Log item associated with the command and choose the `Show Output` item on the Log Item menu. `Show Output` is one of the Browser operations that causes an output window to appear.

Figure 1-44 *Browser: Output Window*



```
Output from Acquire
% acquire -c queries.child queries :example_comp:bin:why
rpc.nsed: Comp :example_comp:bin:why has no revisions
Exit 1      acquire -c queries.child queries :example_comp:bin:why
```

For information on how to manipulate the window attributes of an output window, see the *Windows and Window-Based Tools: Beginner's Guide*.

## 1.7 Browser Operations and Menu Items

All of the operations, or menu items available from the Browser are listed below in alphabetical order.

Browser operations provide some of the queries and commands otherwise available from the command-line interface, described in Chapter 4, *Command-Line Interface*.



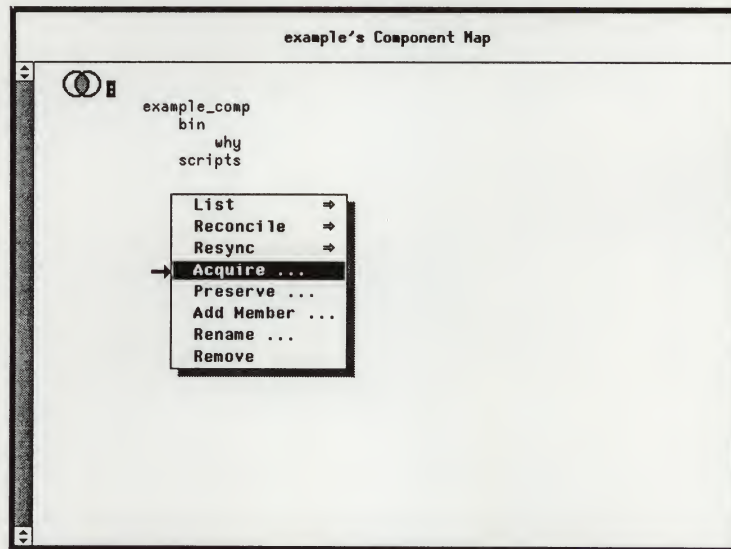
**Acquire...**

The user can access the **Acquire...** operation from either the Component menu or the Display menu.

**Component Menu**

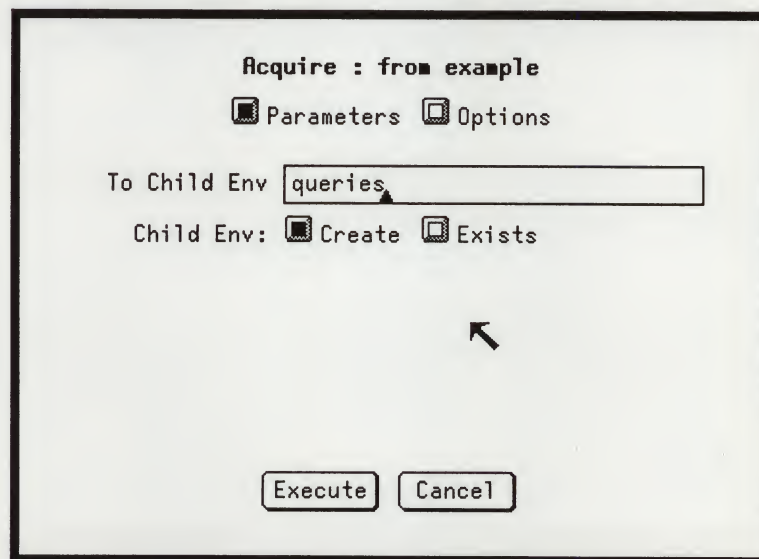
The **Acquire...** item of the Component menu acquires the selected component from the currently activated environment into a child environment.

Figure 1-45 *Browser: Acquire... on Component Menu*



The user specifies the child environment name in a dialog box.

Figure 1-46 *Browser: Acquire... on Component Menu, Dialog Box*



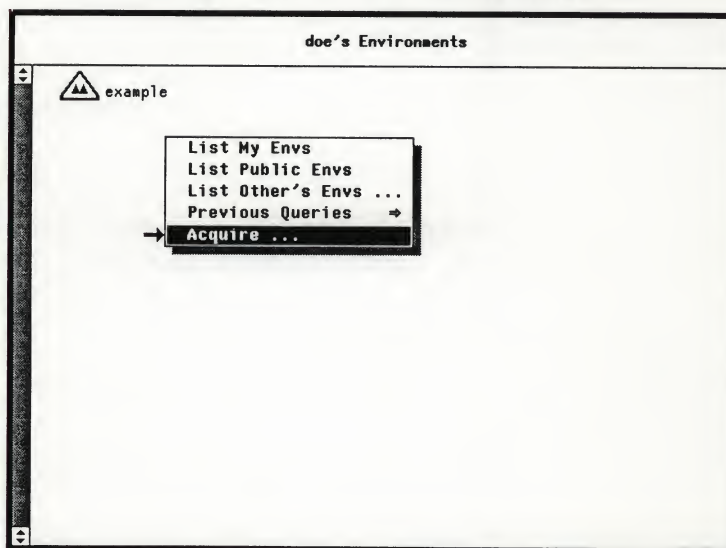
Acquire... fails when the user attempts to acquire a component that has not been preserved. Simply preserve the component using the Preserve... operation and retry the Acquire....

When the administrative user Nse creates an environment using Acquire... into a new child environment, the new environment is a public environment. However, users can't list the newly created public environment until about one hour after the creation of the public environment. The Yellow Pages requires about one hour to propagate information about the new public environment throughout the network.

### Display Menu

The Acquire... item of the Display menu, accessible only when the Display lists environments, acquires a component from one environment to another environment.

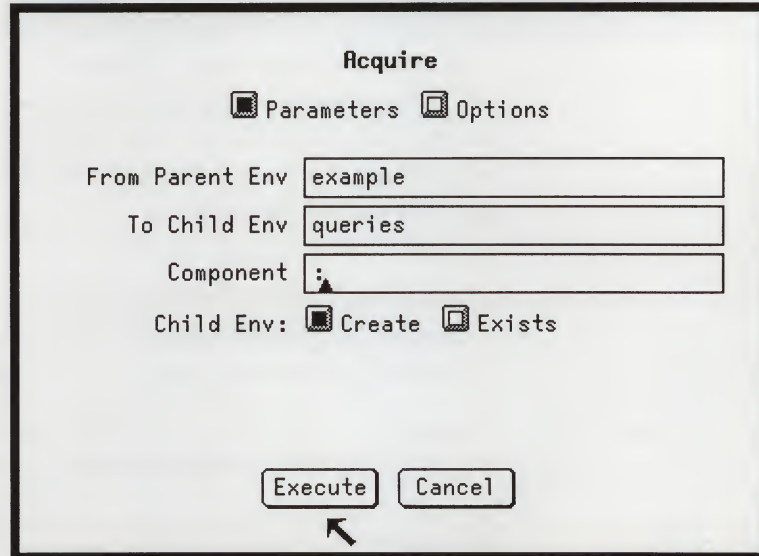
Figure 1-47 *Browser: Acquire... on Display Menu*





The user specifies the source parent environment, the destination child environment, and the name of the component to acquire in a dialog box.

Figure 1-48 *Browser: Acquire... on Display Menu, Dialog Box*



The syntax for component names consists of a root component, specified by a colon ( : ), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

:src:bin:cat

When the user specifies a revision name to the options specification window, *Acquire...* obtains a “consistent” instance, in other words that named revision of the component.

When the user specifies the abbreviation colon ( : ) as the *component* argument to *Acquire...*, the command acquires all of the components from the specified environment

When the user specifies an environment argument to *Acquire...*, *Acquire...* returns an error if the user specifies a */variant* suffix of the environment name, since the operation applies to all variants of the environment.

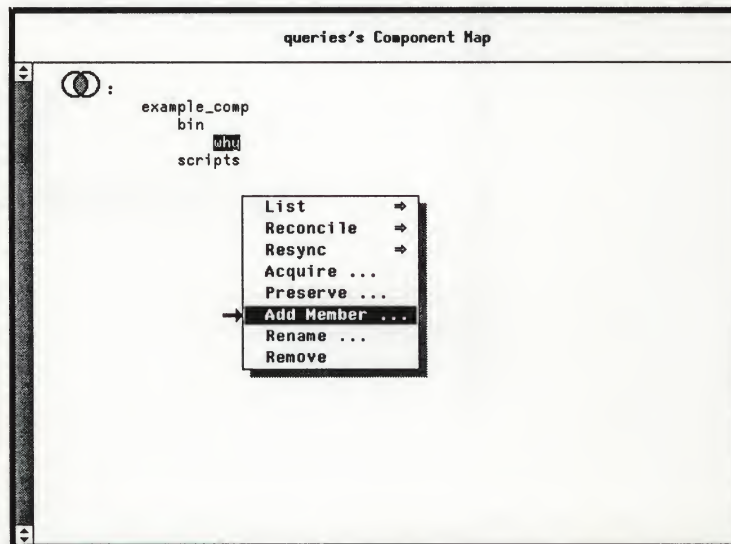
## Add Member...

The Add Member... operation is an item of either the Component menu or the Display menu.

## Component Menu

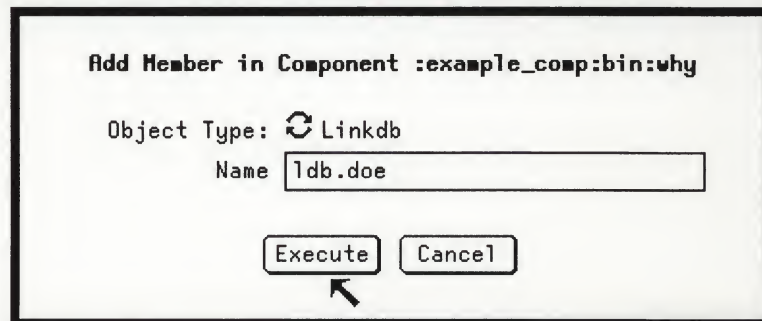
In the Component menu, Add Member... adds an object to the selected component.

Figure 1-49 *Browser: Add Member...*



The user specifies the name and type of the object in a dialog box.

Figure 1-50 *Browser: The Add Member... Dialog Box*





## Display Menu

In the Display menu, Add Member . . . adds an object to the current component.

Figure 1-51 *Browser: Add Member . . .*



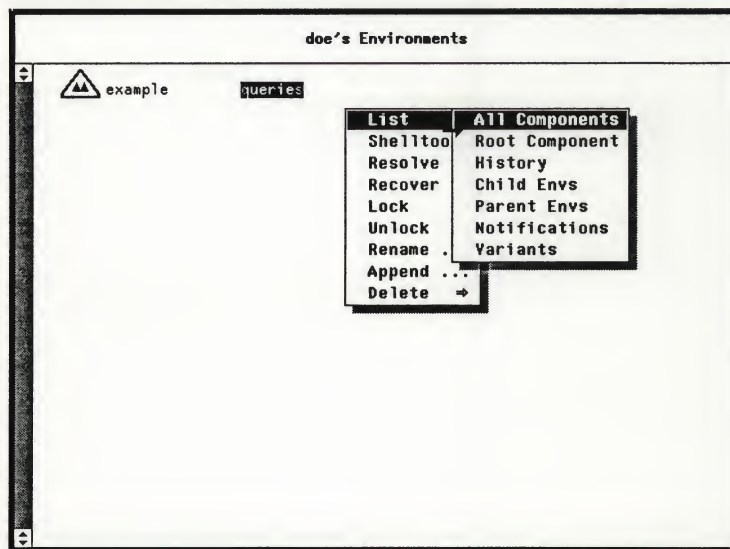
The user specifies the name and type of the object in a dialog box similar to the Add Member . . . dialog box accessible from the Component menu.

## All Components

The user can access the All Components operation from the List => item of either the Environment menu or the Variant menu.

The List => All Components operation of the Environment or Variant menu lists the component hierarchy within the currently selected environment or variant. When the user invokes All Components within the Environment menu, it activates the selected environment with the default variant. When the user invokes All Components within the Variant menu, it activates the selected variant of the current component.

Figure 1-52 *Browser: All Components*



The display of the component hierarchy attempts to convey the structure of the hierarchy by indenting child components about four spaces with respect to the listing of the parent components. When a component has multiple child components and no grandchild components, All Components displays the list of child components in columns. The component map displays only components, not objects of other types. The user can list individual components to find the objects contained in them.



## Append...

The user can access the Append... operation from either the Environment menu or the Variant menu.

The Append... item of the Environment or Variant menu appends an object to the selected environment or variant.

In most cases, the user should choose the Acquire... operation instead of Append... For an explanation, see the description of the acquire command in Chapter 4, *Command-Line Interface*. Append... in the Variant menu is primarily for appending variant-specific derived files.

Figure 1-53 Browser: Append... on Environment Menu



The user specifies the source environment, type of the object, and object name in a dialog box. For syntax information, see Chapter 5, *NSE Naming Specifications*.

Figure 1-54 *Browser: Append... on Environment Menu, Dialog Box*

**Append to environment queries**

☒ Parameters ☒ Options

From Parent Env

Object Type: ☒ Comp

Object Name



## Checkin...

The user can access the Checkin... operation from either the File menu or the Link Database menu.

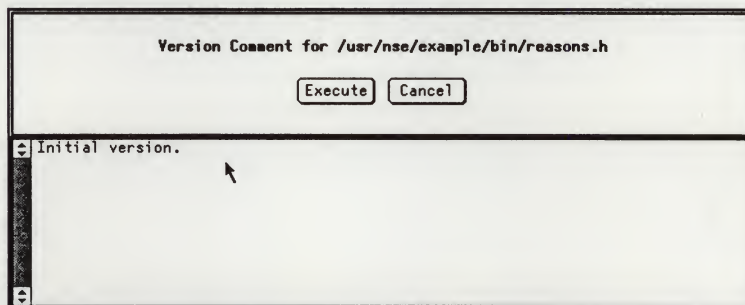
The Checkin... item checks in the selected file or link database to the Version Control System, storing the changes the developer made to that file or link database. Checking in the selected file or link database makes the changes made to the file or link database visible in all of the variants associated with the current environment.

Figure 1-55 *Browser: Checkin... on File Menu*



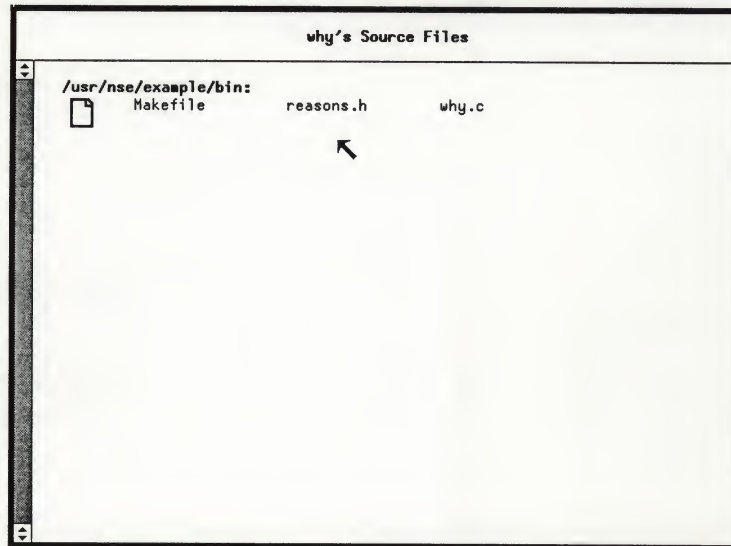
The user specifies a version comment in a dialog box.

Figure 1-56 *Browser: Checkin... on File Menu, Dialog Box*



When a user checks out a file or link database from the Version Control System, the Display puts a check mark in front of the name of the file or link database. Checking in the file, or unchecking out the file, removes the check mark from in front of the filename.

Figure 1-57 *Browser: Checkin... on File Menu, After Operation*



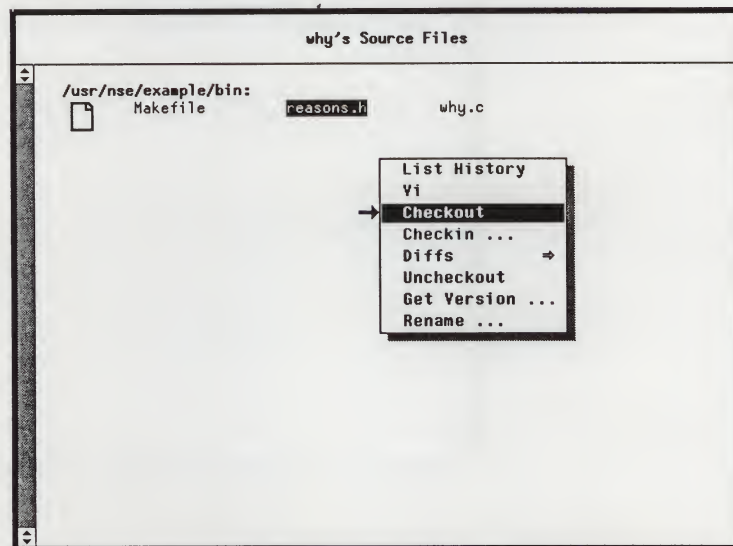


## Checkout

The user can access the `Checkout` operation from either the File menu or the Link Database menu.

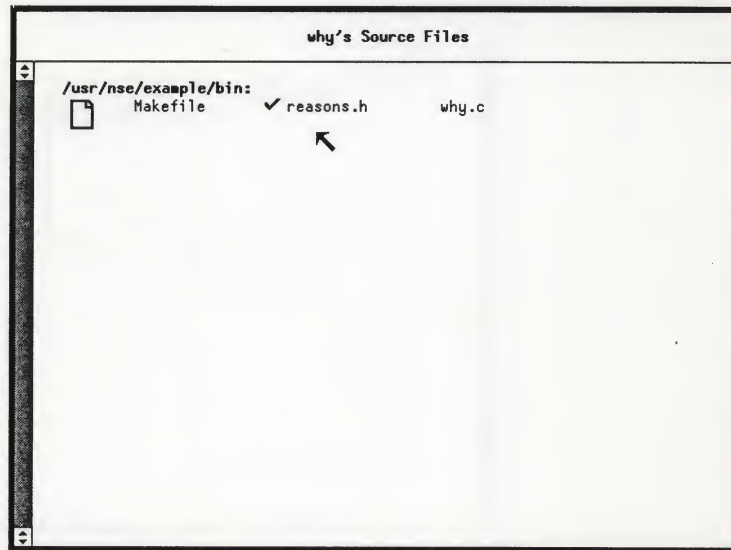
The `Checkout` item checks out the selected file or link database from the Version Control System, so that the user may edit the file or link database.

Figure 1-58 *Browser: Checkout on File Menu*



When a user checks out a file or link database from the Version Control System, the Display puts a check mark in front of the name of the file or link database. Checking in, or unchecking out the file or link database, removes the check mark from in front of the name of the file or link database.

Figure 1-59 *Browser: Checkout on File Menu, After Operation*

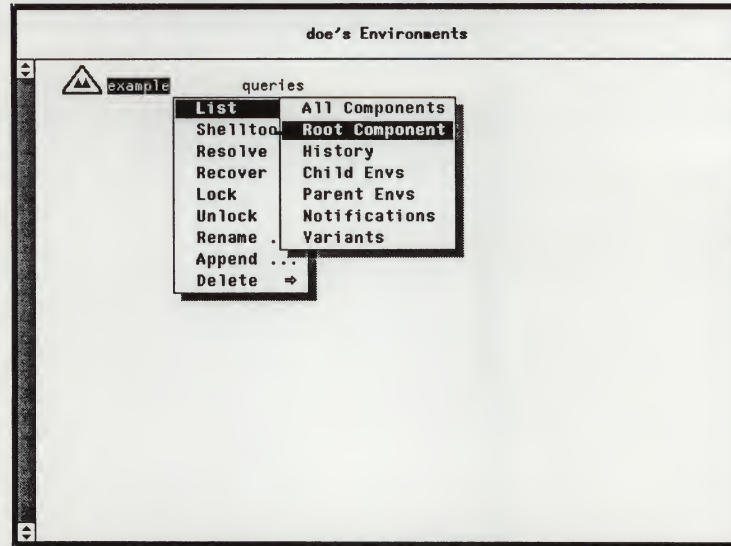




## Child Envs

The user can access the `Child Envs` operation through the `List =>` pull-right menu of the Environment menu. `List =>` `Child Envs` lists, in the Display, the names of the child environments of the selected environment.

Figure 1-60 *Browser: Child Envs*

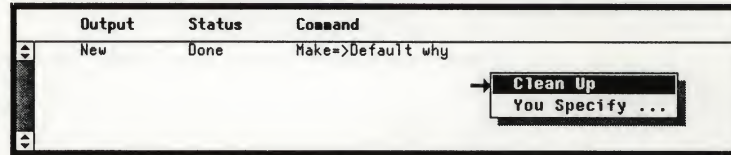


The `Acquire...` and `Append...` operations establish a parent / child relationship between environments. One environment is a child of another environment when a user acquires at least one of its components from the other environment.

## Clean Up

The Clean Up operation is an item of the Log menu. Clean Up clears from the Log all of the log items that have completed execution.

Figure 1-61 *Browser: Clean Up*

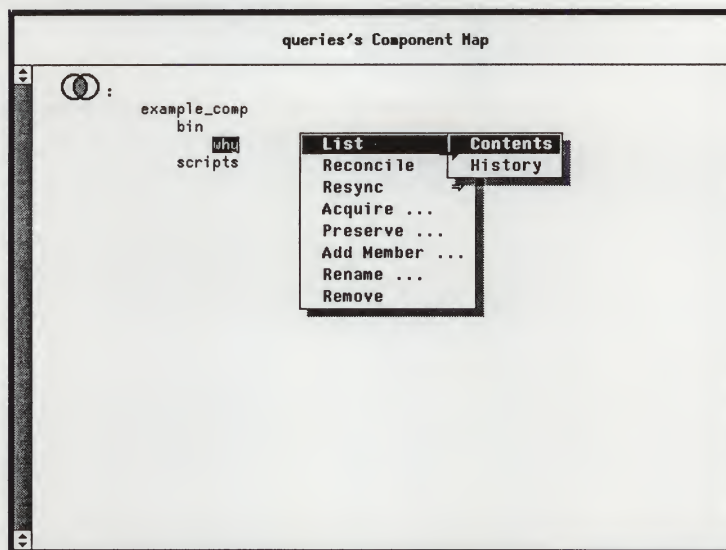




## Contents

The user can access the Contents operation through the List => pull-right of the Component menu. List => Contents lists the objects contained within the selected component.

Figure 1-62 Browser: Contents



The standard NSE objects types are: Comp, Targ, File, and Linkdb.

## Dbxtool

The Dbxtool operation is an item of the Target menu. Dbxtool invokes the dbxtool debugger on the contents of the selected make target. The user should only invoke Dbxtool on targets that represent an executable file, or Dbxtool reports an error.

Figure 1-63 Browser: Dbxtool

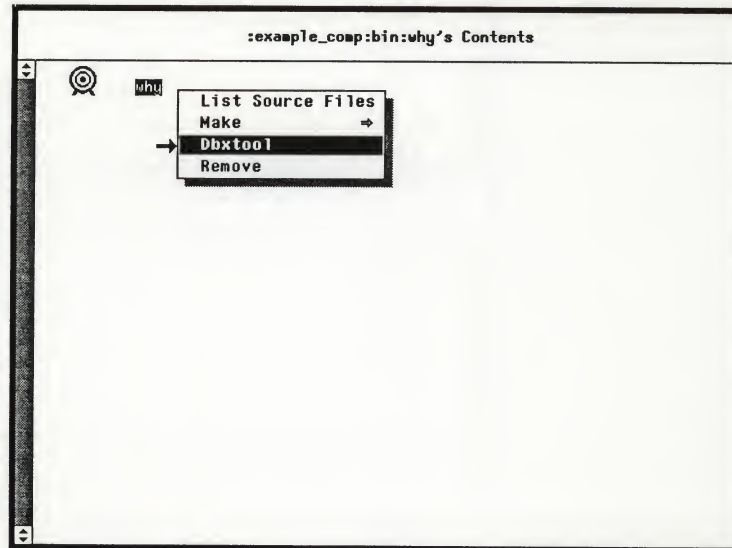
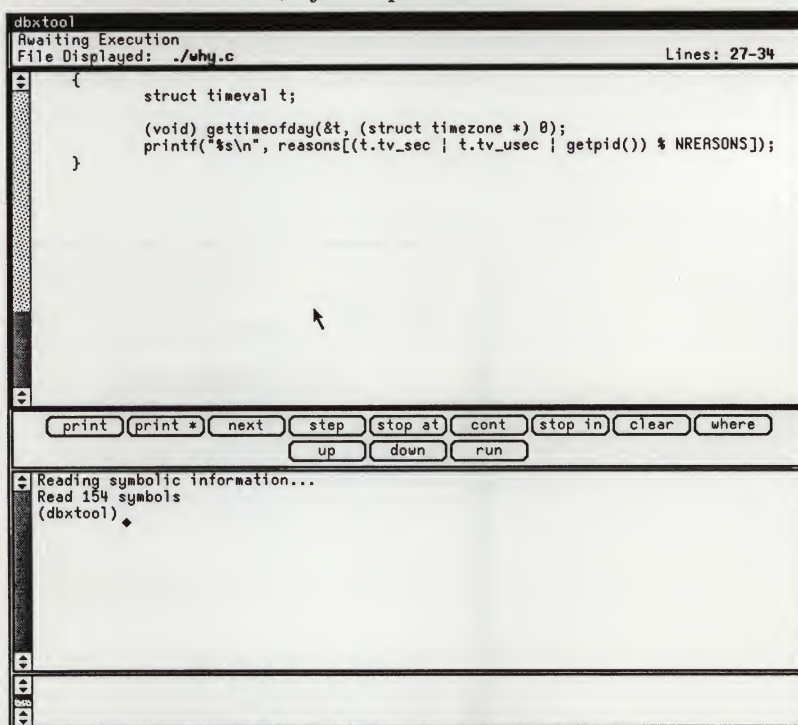




Figure 1-64 *Browser: Dbxtool, After Operation*

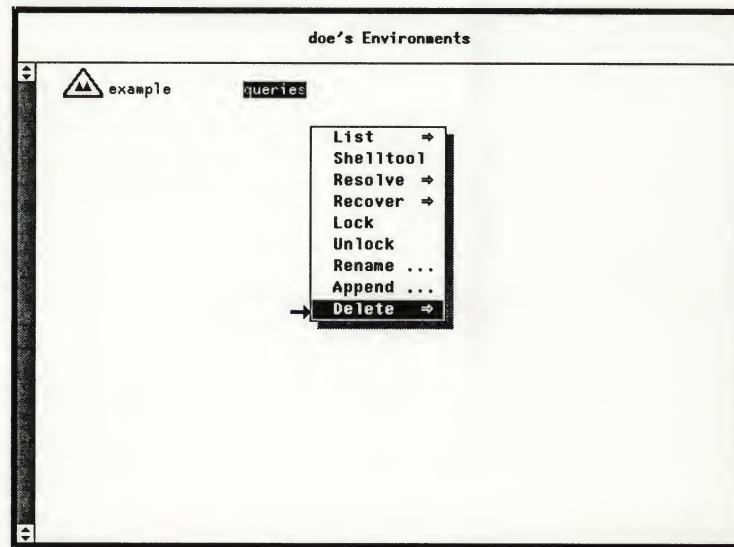
For more information about dbxtool, see the *Programmer's Guide to the Sun Workstation* manual.

Delete =>

The Delete => operation is an item of the Environment menu. Delete => deletes the selected environment. Delete => pulls right to a menu with the following items:

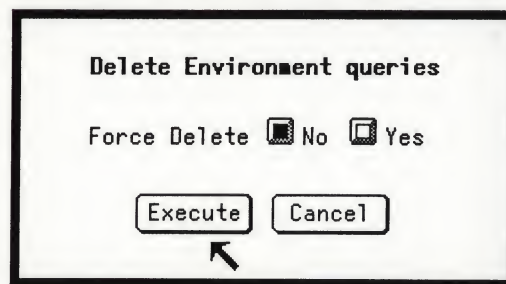
- |            |                                                                  |
|------------|------------------------------------------------------------------|
| Default    | Simply deletes the selected environment.                         |
| Options... | Causes a popup window to appear so the user can specify options. |

Figure 1-65 *Browser: Delete =>*



The Options... item on the Delete => pull-right menu causes a popup window to appear.

Figure 1-66 *Browser: Delete => Options..., Popup Window*



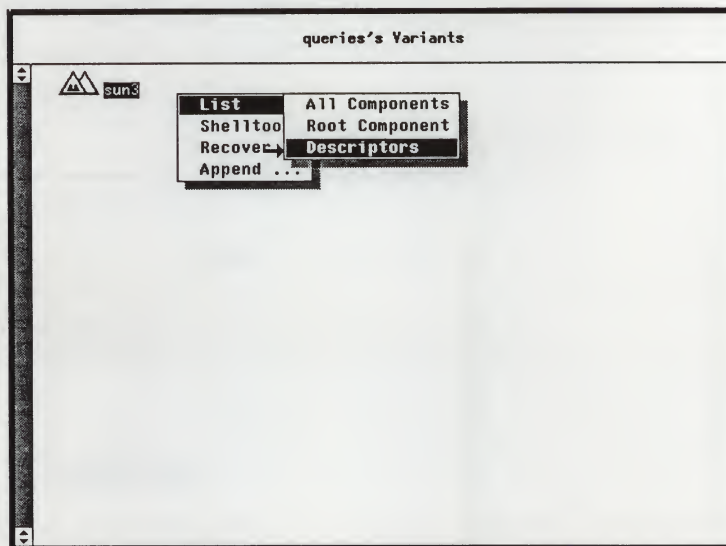
The one option to the Delete => operation controls whether or not the user can delete the selected environment when that environment has children.



## Descriptors

The user can access the Descriptors operation through the List => item of the Variant menu.

Figure 1-67 *Browser: Descriptors*



Descriptors lists the variant descriptors for the selected variant.

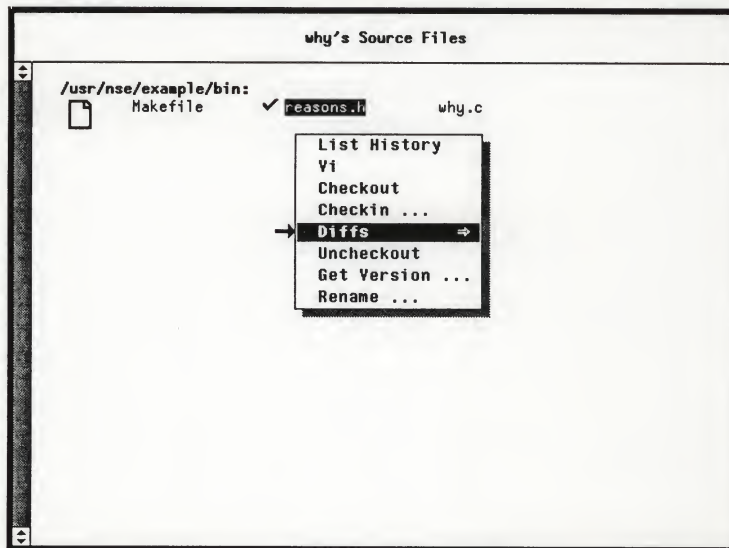
Figure 1-68 *Browser: Descriptors, Descriptive Information*



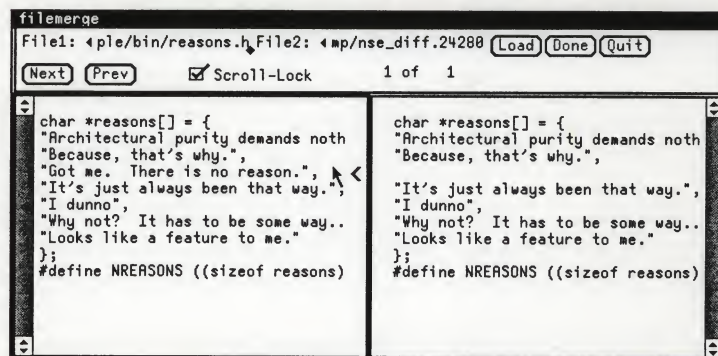
Variant descriptors are a series of shell environment variables and the values the NSE assigns to the variables when the user activates the selected variant.

Diffs =&gt;

The Diffs => operation is an item of the File menu.  
Diffs => pulls right to a menu with the following items:  
Latest, Version..., and Two Versions....

Figure 1-69 *Browser: Diffs =>*

Choosing the Latest item compares, using the filemerge application, the current working version of the selected file with the most recent checked in version of the file.

Figure 1-70 *Browser: Diffs => Latest and filemerge*



Choosing the Version... item of the Diffs => pull-right compares, using the filemerge application, the current working version of the selected file with a version the user specifies in a dialog box window.

Choosing the Two Versions... item compares, using the filemerge application, one version of a file the user specifies with another version of a file the user specifies in a dialog box window that appears.

## Dismiss

The `Dismiss` operation is an item of the Log Item menu. `Dismiss` clears the selected log item from the list of log items in the Log. `Dismiss` only clears items that have completed execution.



*(Editor)*

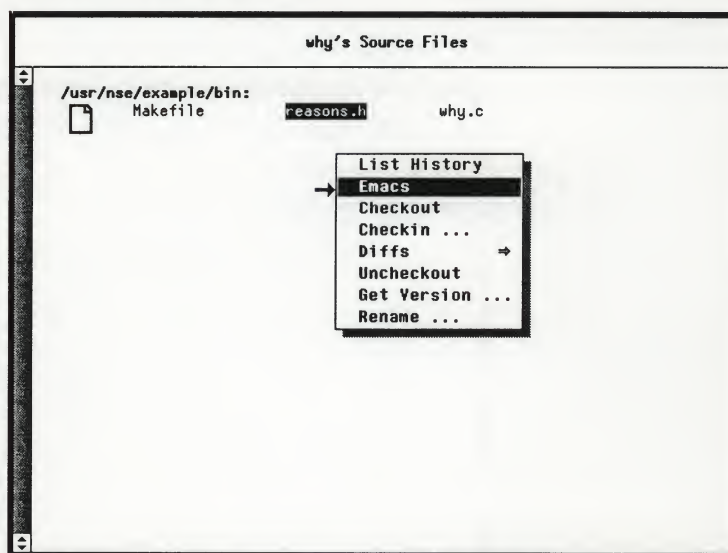
The editor operation is an item of the File menu . The default menu item is `Vi`, but the user may specify `Emacs` or `Textedit` as an alternate menu item. In any case, this editor menu item causes a window to appear that executes the editor on the selected file.

For information on any of the three editor operations mentioned above, look within this section. For information on any of the three editors, see the Man Page for that editor, online or in the *Commands Reference Manual*. For information on how to change the editor that this item invokes, see Section 1.8, *Using a Text Editor with the Browser*.

## Emacs

The Emacs operation may be an item of the File menu. The default editor item for the File menu is Vi, but the user may specify Emacs or Textedit as an alternate menu item. For information on how to change the editor item so that it invokes the Emacs operation, see Section 1.8, *Using a Text Editor with the Browser*.

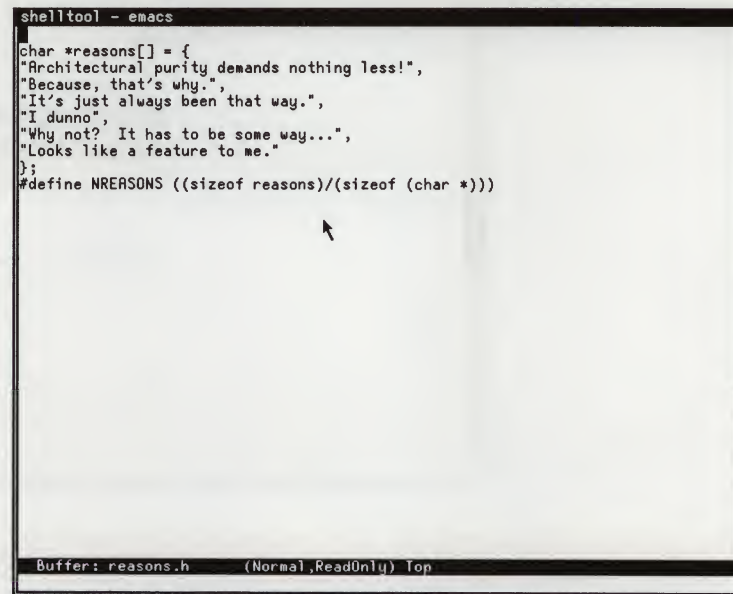
Figure 1-71 Browser: Emacs





The first time the user invokes the Emacs operation, it causes a window to appear that executes the editor on the selected file. Subsequent Emacs operations load the selected file into the same Emacs, if it still exists. However, the Emacs operation starts a new Emacs when the user selects a file from another environment and invokes the Emacs operation. Since each environment presents a different view of the file system, each environment must have its own Emacs.

Figure 1-72 *Browser: Emacs, Popup Window*



```
shelltool - emacs
char *reasons[] = {
    "Architectural purity demands nothing less!",
    "Because, that's why.",
    "It's just always been that way.",
    "I dunno",
    "Why not? It has to be some way...",
    "Looks like a feature to me."
};
#define NREASONS ((sizeof reasons)/(sizeof (char *)))

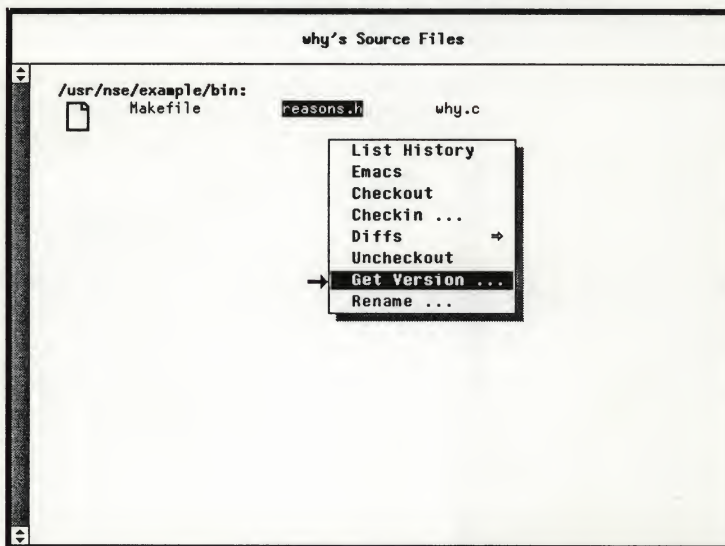
Buffer: reasons.h (Normal,ReadOnly) Top
```

## Get Version...

The user can access the Get Version... operation on either the File menu or the Link Database menu.

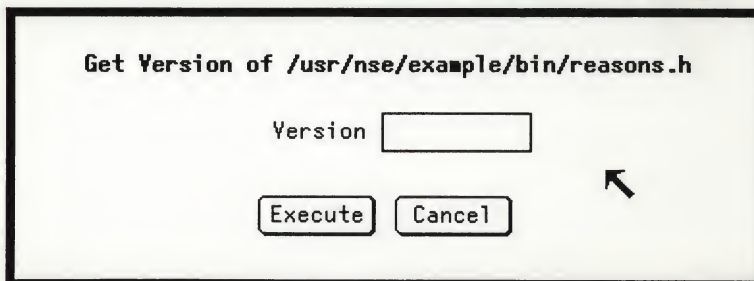
The Get Version... item obtains a named version of the selected file or link database.

Figure 1-73 *Browser: Get Version... on File Menu*



The user specifies the version name in a dialog box.

Figure 1-74 *Browser: Get Version... on File Menu, Dialog Box*



The user can obtain the possible file version names from the List History operation output for the selected file.



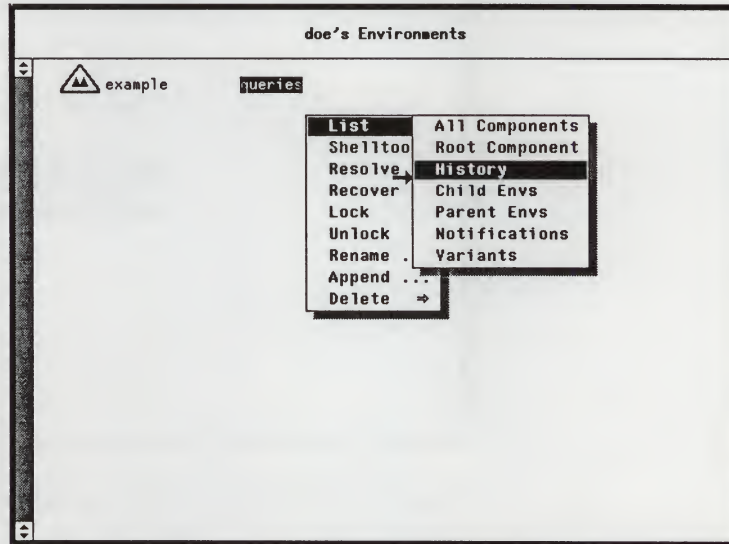
## History

The user can access the History operation through the List => pull-right menu of either the Environment menu or the Component menu.

## Environment Menu

The List => History item of the Environment menu lists all the revisions for all of the components within the selected environment.

Figure 1-75 *Browser: History on Environment Menu*



Choosing the `History` item causes a popup window to appear which lists the revision history for all of the components in the selected environment.

Figure 1-76 *Browser: History on Environment Menu, Popup Window*



This revision history information includes the following information for each revision of the selected component:

- The revision number and possibly its name.
- The time of the revision.
- The comments associated with that revision.
- The username of the user who created the revision.
- The names of the components the revision applies to.

#### Component Menu

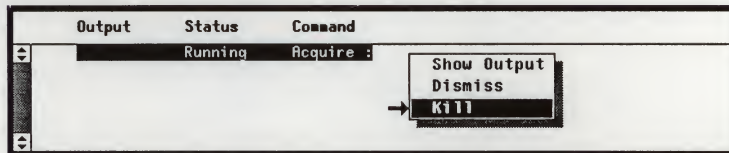
The `List => History` item of the Component menu lists the revision history of the selected component.



## Kill

The `Kill` operation is an item of the Log Item menu. `Kill` aborts the execution of the command represented by the selected item in the Log.

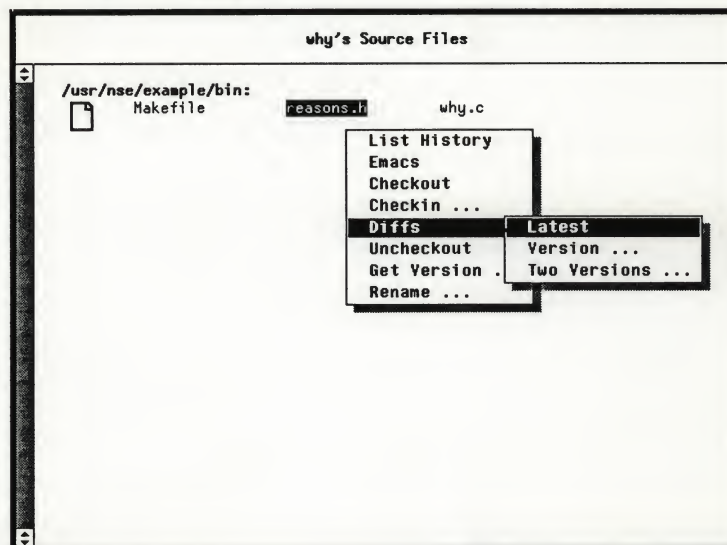
Figure 1-77 *Browser: Kill*



## Latest

The user can access the Latest operation through the Diffs => item of the File menu. Diffs => Latest compares the current working version of the selected file with the latest version of the file. See the Diffs => operation for more information.

Figure 1-78 Browser: Latest





## List =&gt;

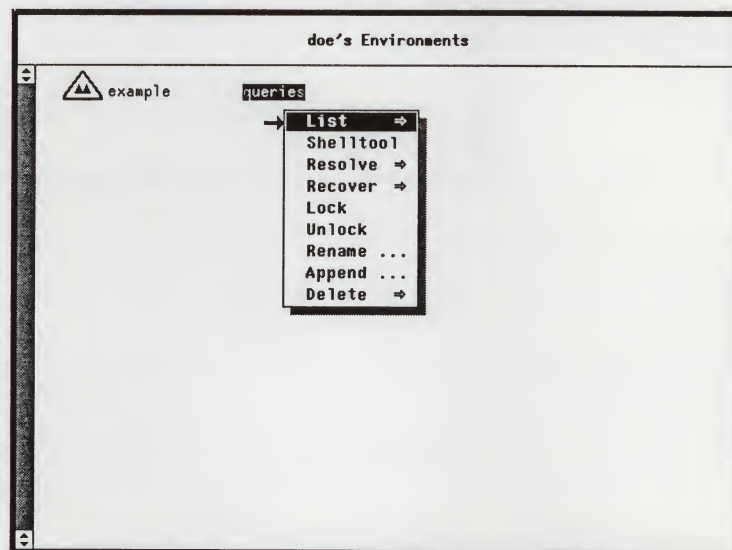
The List => operation is an item on the Environment menu, the Component menu, and the Variant menu. Choosing the List => item lists one of the items the user chooses from the pull-right menu.

## Environment Menu

The List => item of the Environment menu pulls right to the following menu items:

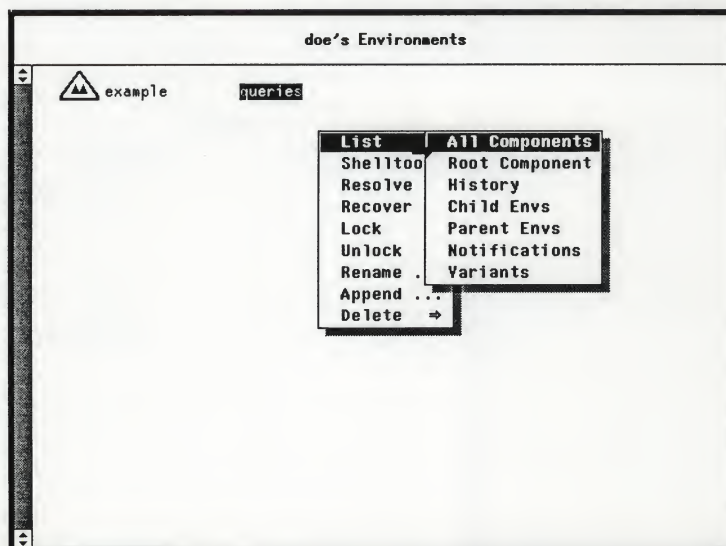
- All Components
- Root Component
- History
- Child Envs
- Parent Envs
- Notifications
- Variants

Figure 1-79 *Browser: List => on Environment Menu*



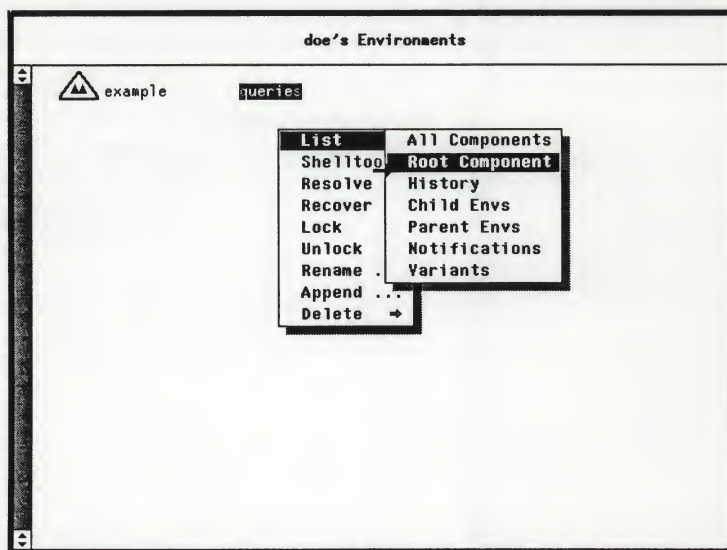
The List => All Components item lists the component hierarchy of the selected environment in the Display.

Figure 1-80 Browser: List => All Components on Environment Menu



The List => Root Component item lists the contents of the root component of the selected environment.

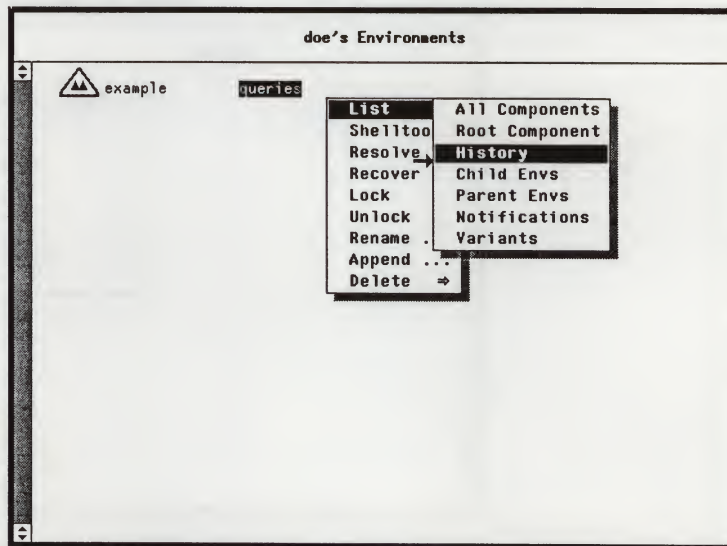
Figure 1-81 Browser: List => Root Component on Environment Menu





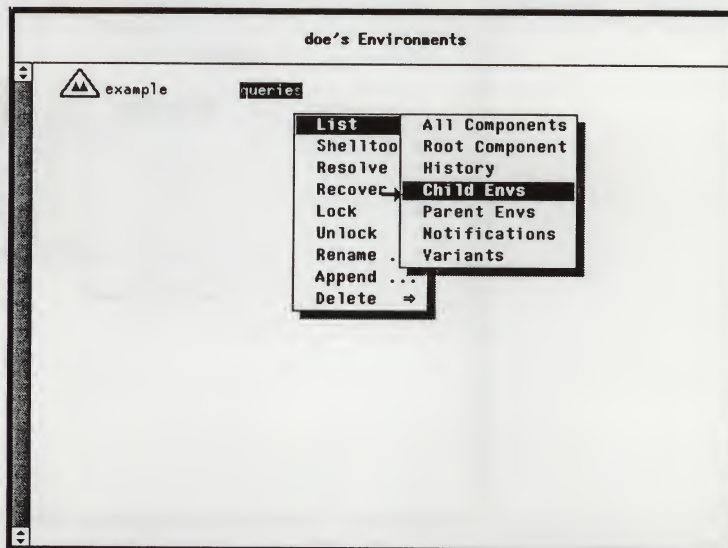
The List => History item causes a popup window to appear which lists the history information associated with the selected environment.

Figure 1-82 *Browser: List => History on Environment Menu*



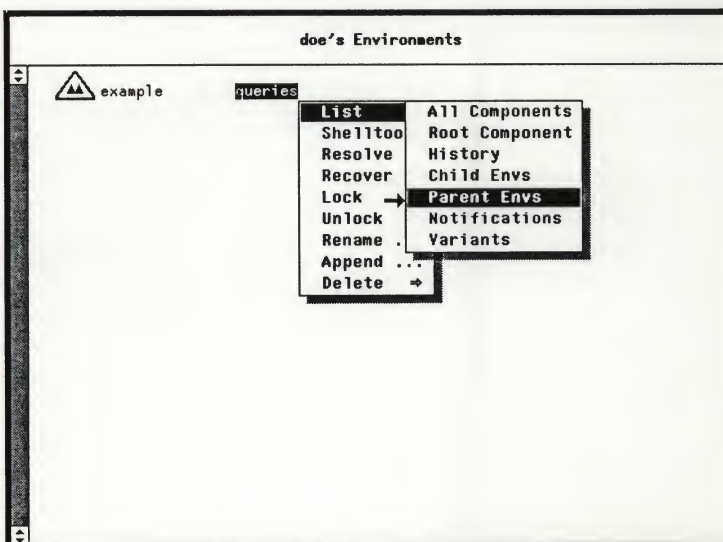
The List => Child Envs item lists the names of the child environments of the selected environment.

Figure 1-83 *Browser: List => Child Envs on Environment Menu*



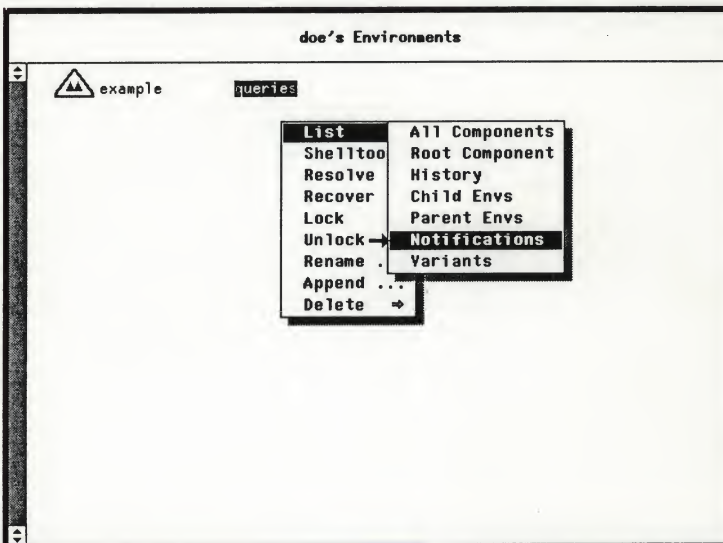
The List => Parent Envs item lists the names of the parent environments of the selected environment.

Figure 1-84 Browser: List => Parent Envs *on Environment Menu*



The List => Notifications item lists the notification requests associated with the selected environment.

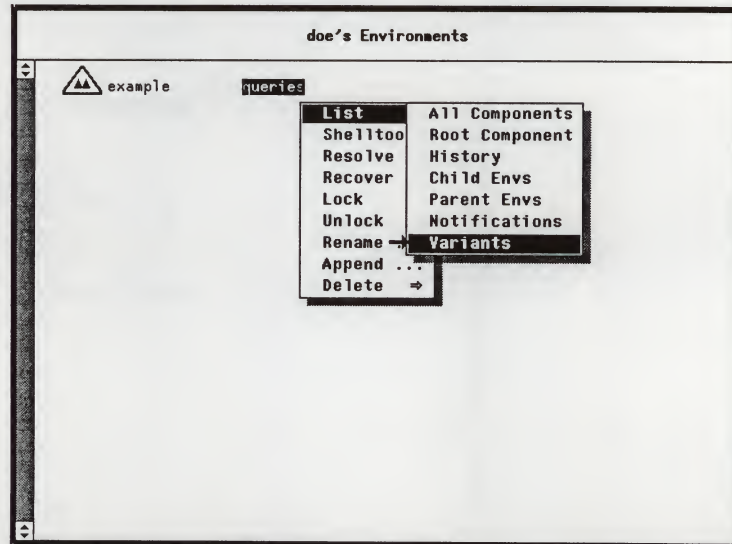
Figure 1-85 Browser: List => Notifications *on Environment Menu*





The List => Variants item lists the names of the variants associated with the selected environment.

Figure 1-86 *Browser: List => Variants on Environment Menu*



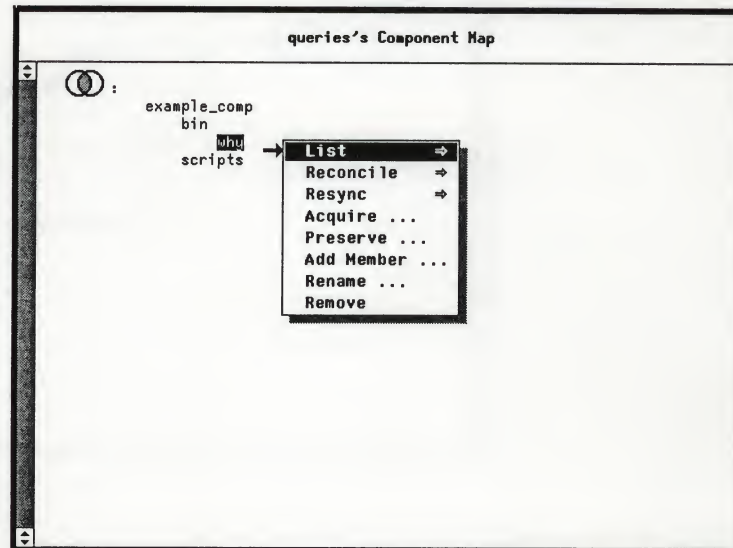
## Component Menu

The List => item of the Component menu pulls right to the following menu items:

Contents

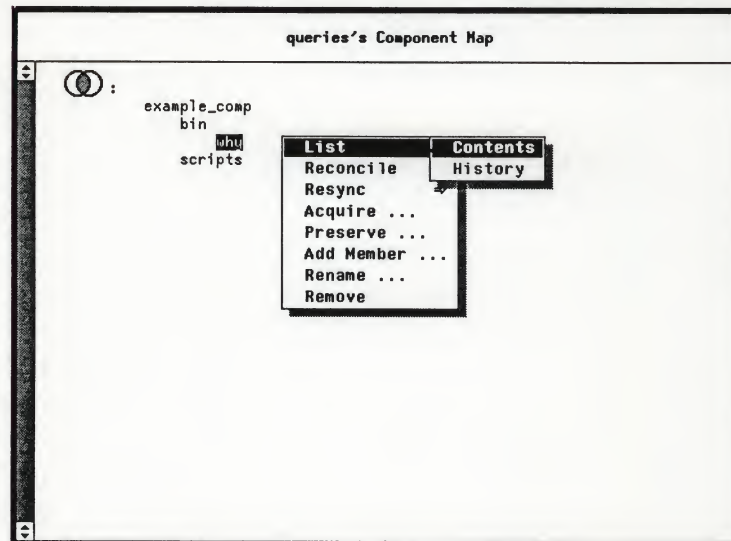
History

Figure 1-87 *Browser: List => on Component Menu*



The List => Contents item lists, in the Display, the objects contained in the selected component.

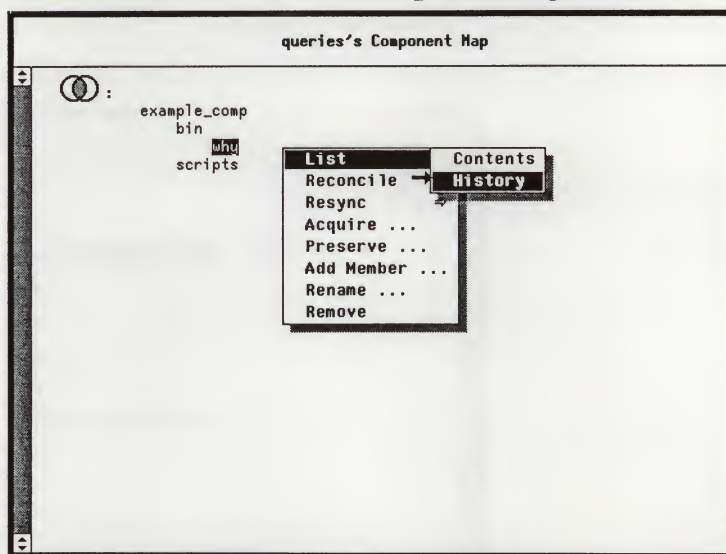
Figure 1-88 *Browser: List => Contents on Component Menu*





The List => History item lists, in the Display, the revision history of the selected component.

Figure 1-89 *Browser: List => History on Component Menu*



## Variant Menu

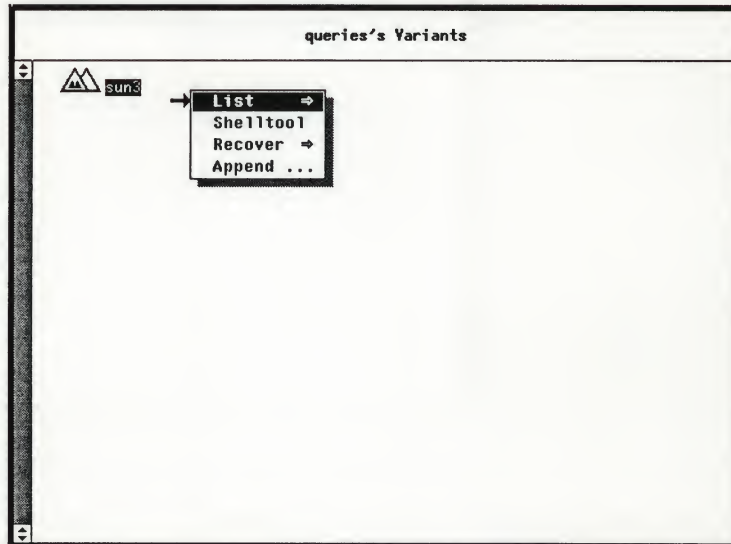
The List => item of the Variant menu pulls right to the following menu items:

All Components

Root Component

Descriptors

Figure 1-90 *Browser: List => on Variant Menu*

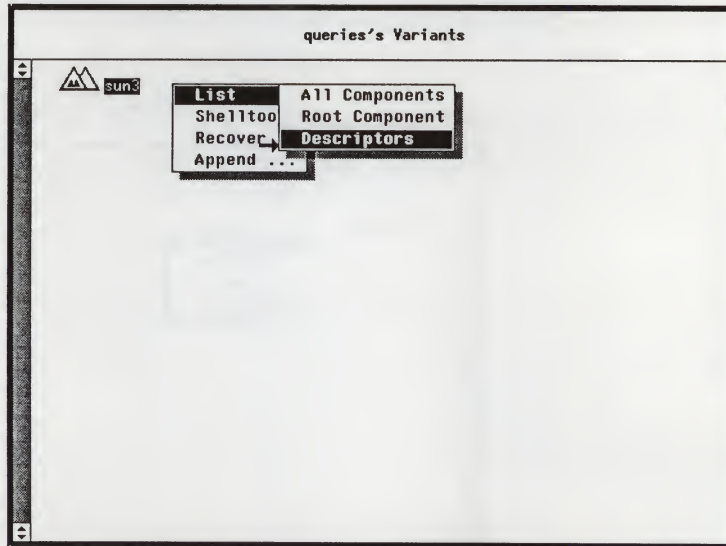


The All Components and Root Component items behave the same way as in the Environment menu.



The List => Descriptors item lists the descriptors of the selected variant.

Figure 1-91 *Browser: List => Descriptors on Variant Menu*

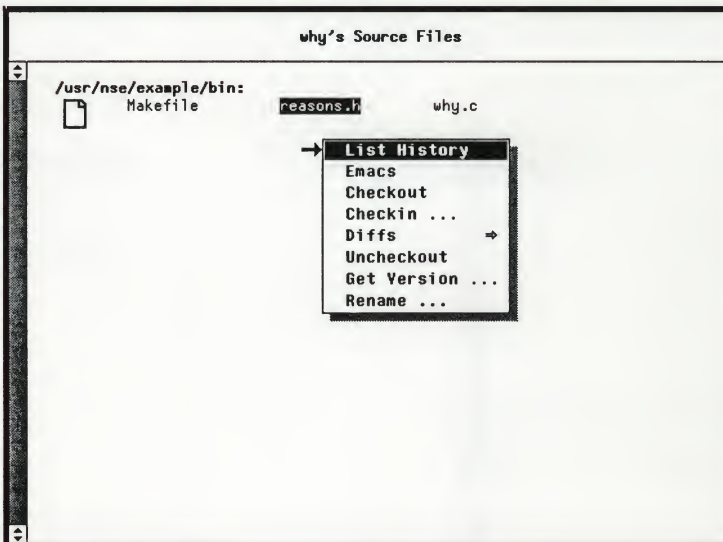


For more information about the descriptors, see the description of the Descriptors operation.

## List History

The List History operation is an item on the File menu. List History lists the Version Control System history for the selected file.

Figure 1-92 *Browser: List History*



List History causes a popup window to appear which lists the Version Control history for the selected file.

This Version Control history information includes the following information for each version of the selected file:

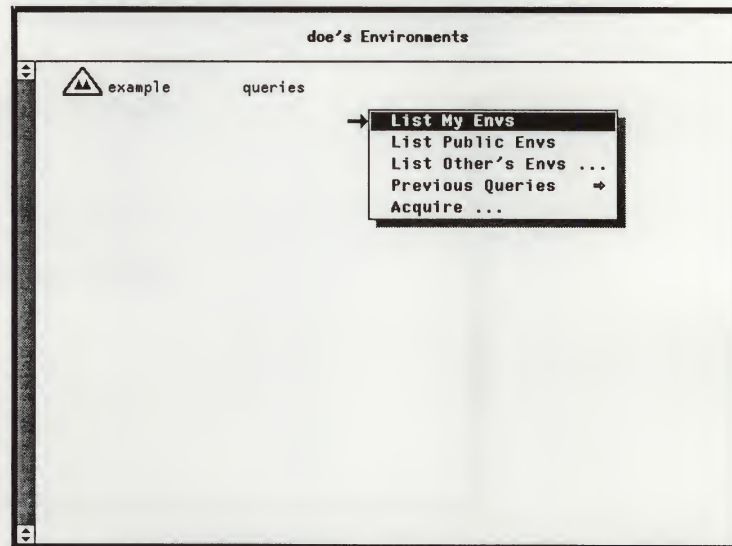
- The version number.
- The time of the version.
- The comments associated with that version.
- The username of the user who created the version.



## List My Envs

The user can access the List My Envs operation on either the Legend menu or the Display menu. List My Envs provides a list, in the Display, of the environments owned by the user who invokes the Browser.

Figure 1-93 *Browser: List My Envs on Display Menu*



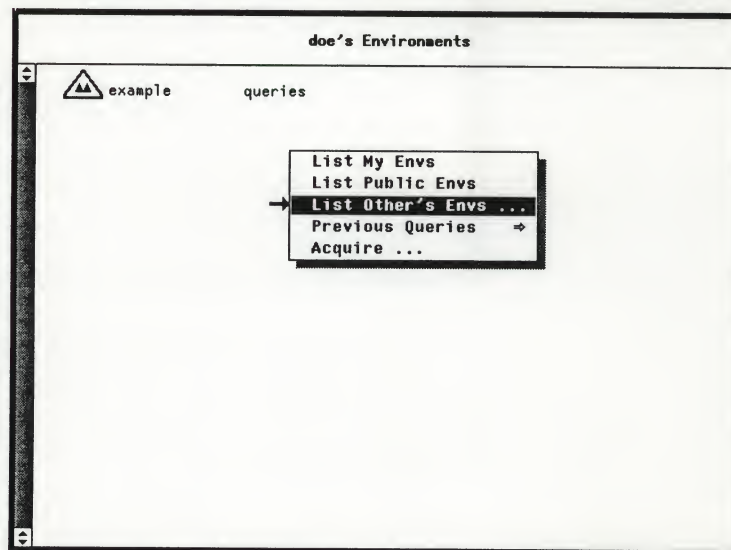
## List Other's Envs...

The user can access the List Other's Envs... operation on either the Legend menu or the Display menu. List Other's Envs... provides a list, in the Display, of the environments owned by another user.

List Other's Envs... won't list the environments of another user unless the user who invokes the command has mounted the home directory of the other user.

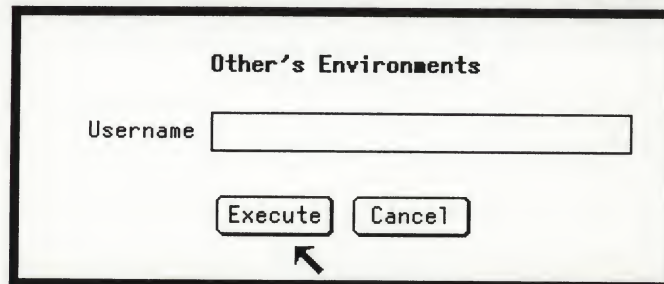
The List Other's Envs... item provides a list, in the Display, of the environments owned by another user.

Figure 1-94 *Browser: List Other's Envs... on Display Menu*



The user specifies the username of the other user in a dialog box.

Figure 1-95 *Browser: List Other's Envs..., Display Menu, Dialog Box*

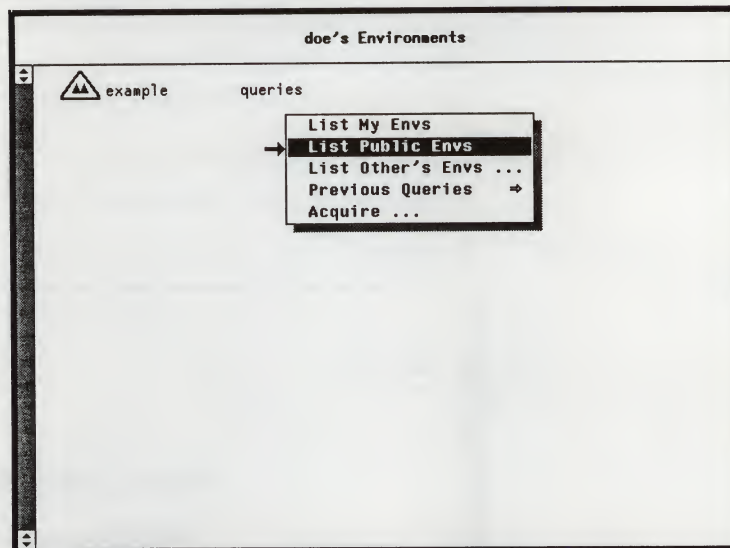




## List Public Envs

The user can access the List Public Envs operation on either the Legend menu or the Display menu. List Public Envs provides a list, in the Display, of the public environments.

Figure 1-96 *Browser: List Public Envs on Display Menu*



Users can't list newly created public environments until about one hour after their creation. The Yellow Pages requires about one hour to propagate information about a new public environment throughout the network.

## List Source Files

The List Source Files operation is an item of the Target menu. List Source Files lists the names of the source files the selected make target contains.

Figure 1-97 *Browser: List Source Files*

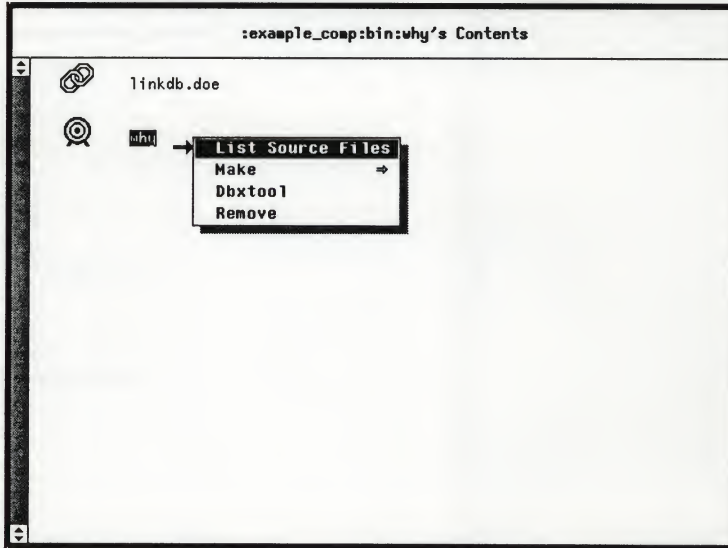
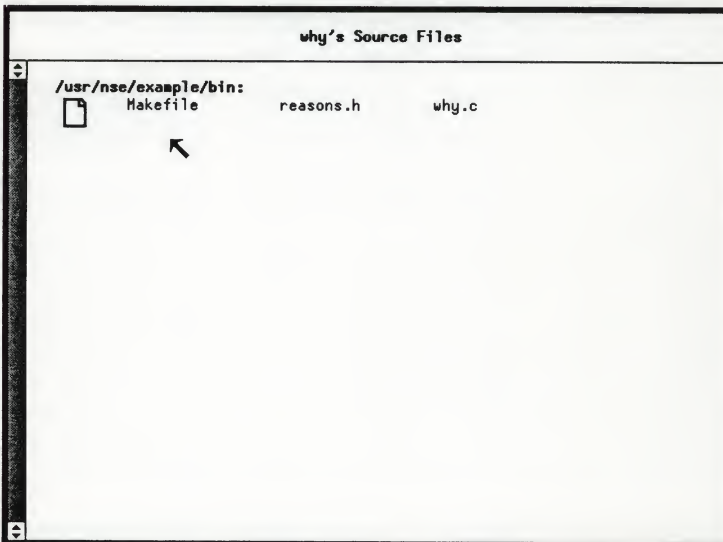


Figure 1-98 *Browser: List Source Files, After Operation*

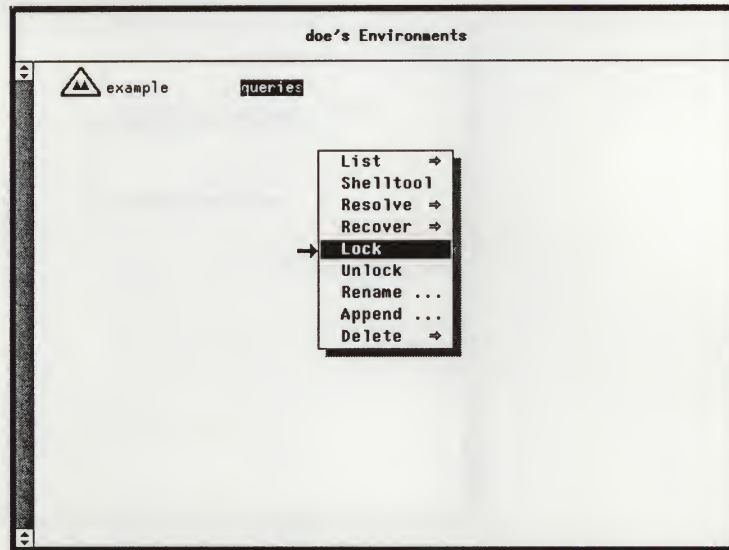




## Lock

The `Lock` operation is an item of the Environment menu. `Lock` locks the selected environment so that no other user can make changes to the environment until someone releases the lock on that environment.

Figure 1-99 *Browser: Lock*



When a user activates an environment that has children, the environment activates permitting reading only. However, when a user obtains a lock for the environment that has children, the user can activate the environment with read and write privileges.

See also the `Unlock` operation.

Make =>

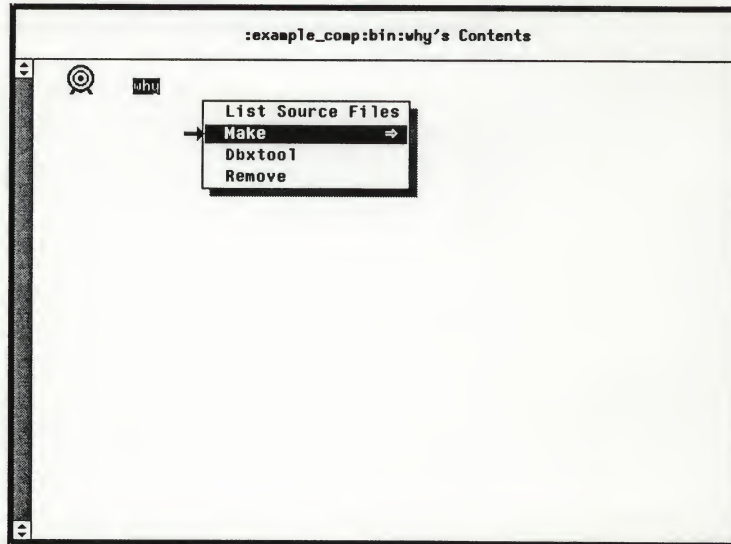
The Make => operation is an item of the Target menu.

Make => builds the make target using the make program.

Make => pulls right to the following menu items:

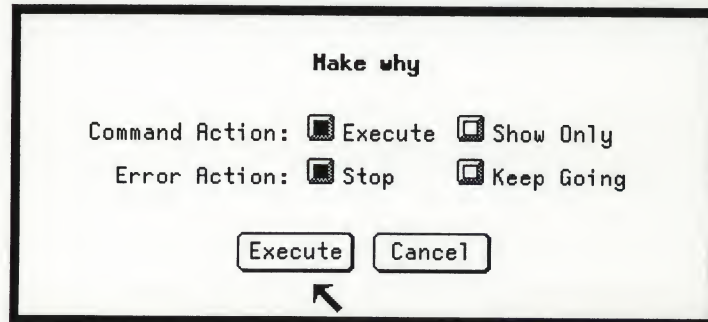
- |                 |                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------|
| Default         | Simply build the selected target.                                                                   |
| Options...      | Cause a popup window to appear so the user can specify options to the build of the selected target. |
| Other Target... | Cause a popup window to appear so the user can specify another target for the build.                |

Figure 1-100 Browser: Make =>



The Options... item on the Make => pull-right menu causes a popup window to appear.

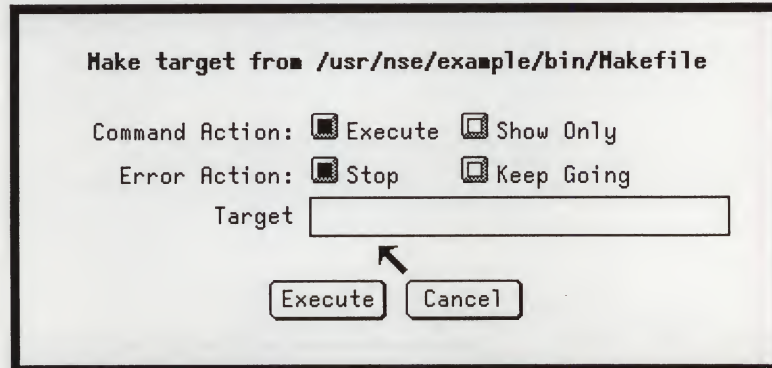
Figure 1-101 Browser: Make => Options..., Popup Window





The Other Target... item on the Make => pull-right menu causes a dialog box to appear. The dialog box permits the user to specify a build of a different target within the same Makefile.

Figure 1-102 Browser: Make => Other Target..., Dialog Box

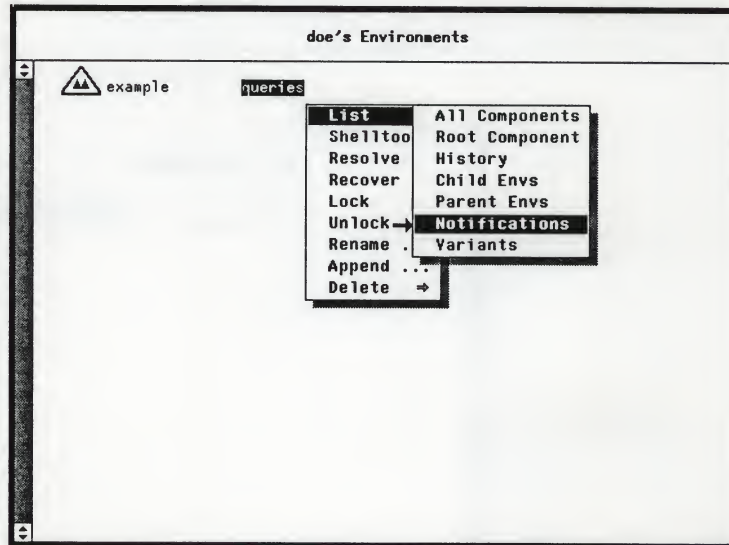


For more information about the make program, see the *Programmer's Guide to the Sun Workstation* or the *make User's Guide* manual, or the *make Man Page*, online or in the *Commands Reference Manual*.

## Notifications

The user can access the Notifications operation through the List => item of the Environment menu. Notifications lists the notification requests associated with the selected environment.

Figure 1-103 Browser: Notifications



The notification facility sorts notification requests by these event types: reconcile-to, acquire-from, resync-from, envlock, and envunlock. Within each of these types, there is a list of requests, which include: object type and name (optional), recipient (*username@machine*), and command to execute (default is Mail).

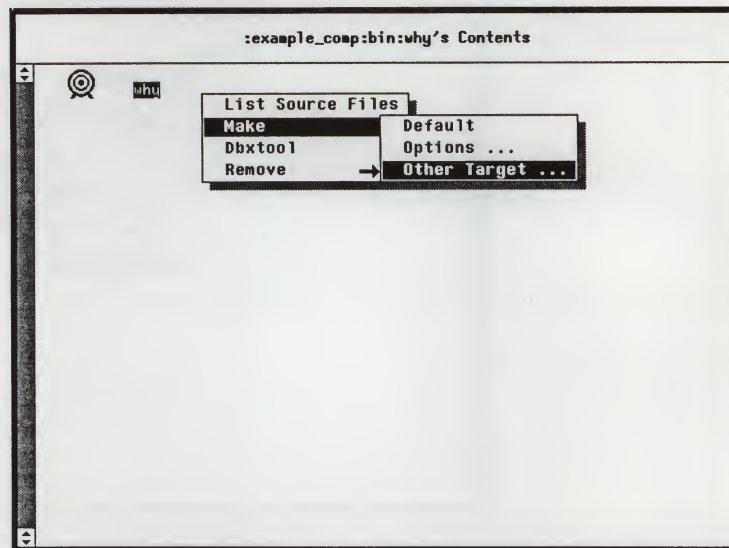
See also the Register Notification... and Unregister operations.



## Other Target...

The user can access the Other Target... operation through the Make => pull-right menu of the Target menu. Other Target... causes a popup window to appear so the user can specify another target within the same Makefile for the build.

Figure 1-104 Browser: Other Target...

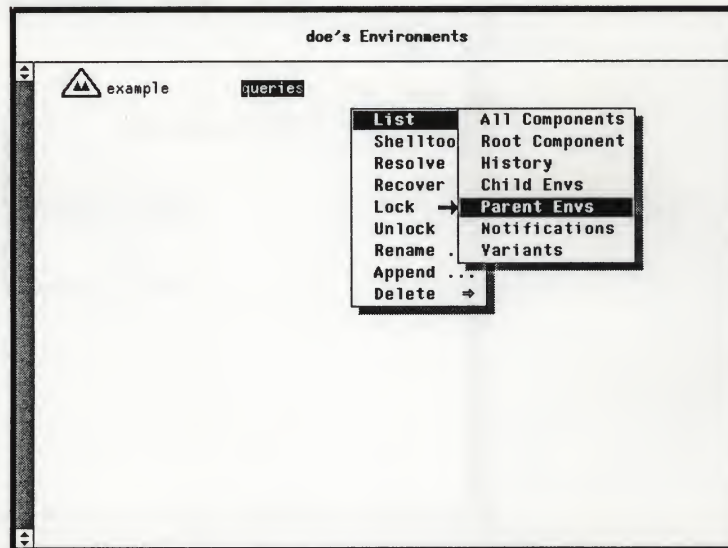


For more information, see the Make => operation.

## Parent Envs

The user can access the Parent Envs operation through the List => pull-right menu of the Environment menu. List => Parent Envs lists, in the Display, the names of the parent environments of the selected environment.

Figure 1-105 Browser: Parent Envs

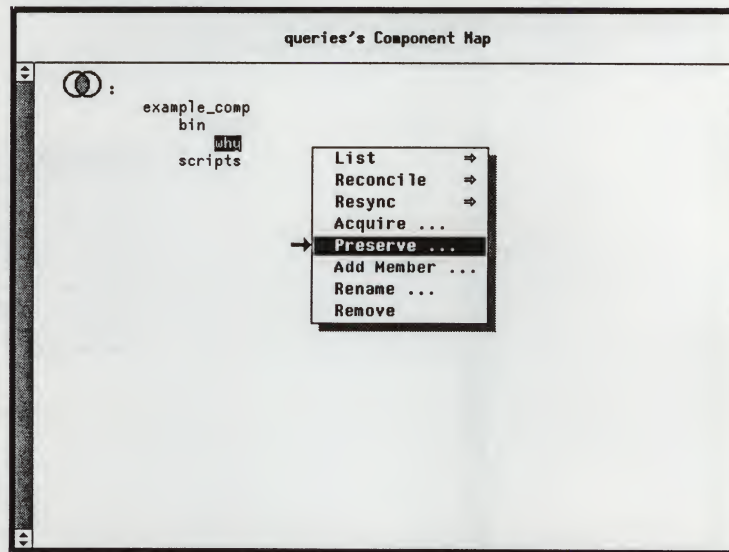




## Preserve...

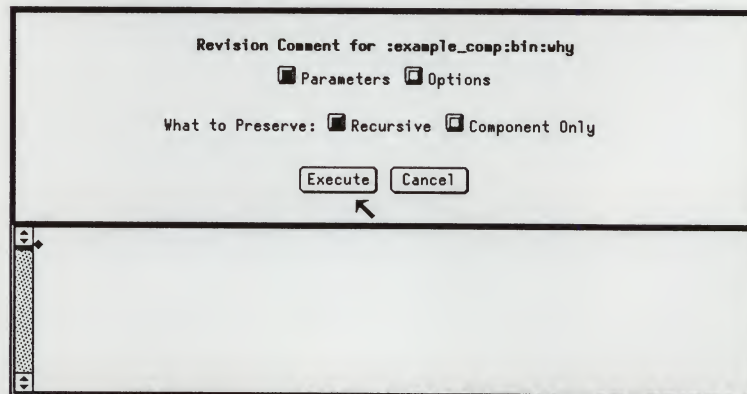
The Preserve... operation is an item of the Component menu. Preserve... makes a revision of the current contents of the selected component in the currently activated environment.

Figure 1-106 *Browser: Preserve...*



The user specifies the name of the revision, some revision comments, and whether or not to preserve only the top-level component in a dialog box.

Figure 1-107 *Browser: Preserve..., Dialog Box*

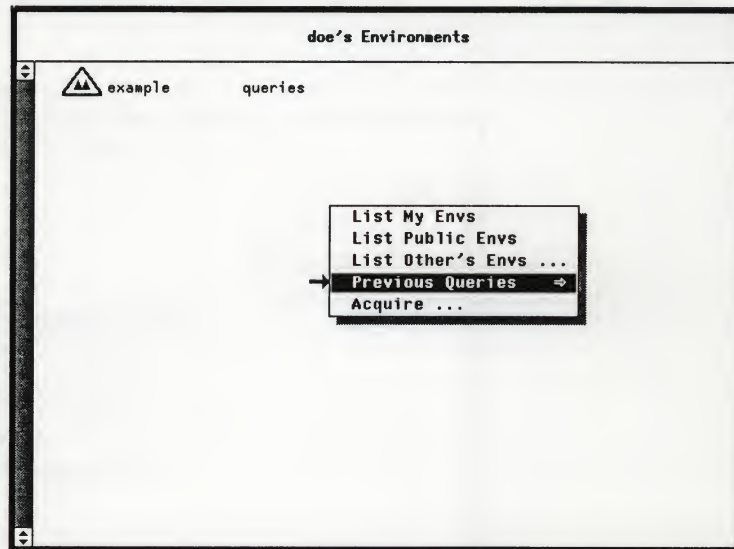


When Preserve... provides an error stating that the objects it is preserving are not "stable," the user can try using the Checkin... operation on those objects to make them stable.

## Previous Queries =&gt;

The user can access the Previous Queries => operation from either the Legend menu or the Display menu. In both cases, Previous Queries => pulls right to a list of up to ten of the most recent browsing operations the user invoked in the Browser. By choosing one of the previous queries, the user can return the Browser to a previous display.

Figure 1-108 *Browser: Previous Queries =>*





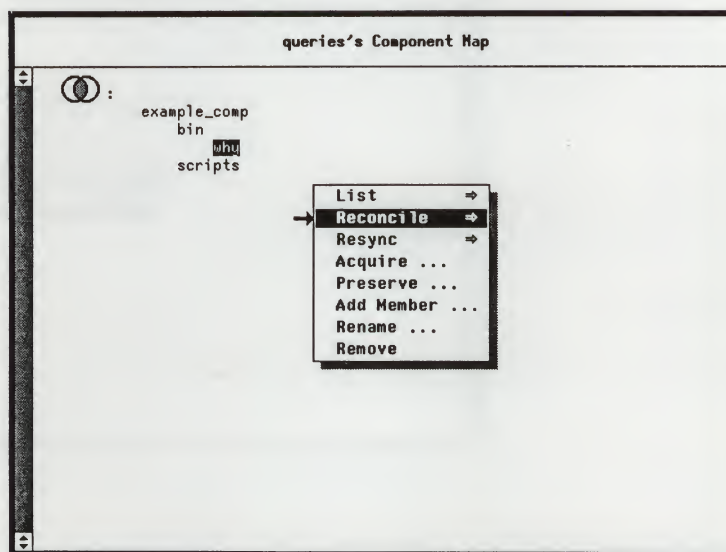
## Reconcile =&gt;

The Reconcile => operation is an item of the Component menu. Reconcile => reconciles the selected component in the currently activated environment to its parent environment.

Reconcile => item of the Component menu pulls right to the following menu items:

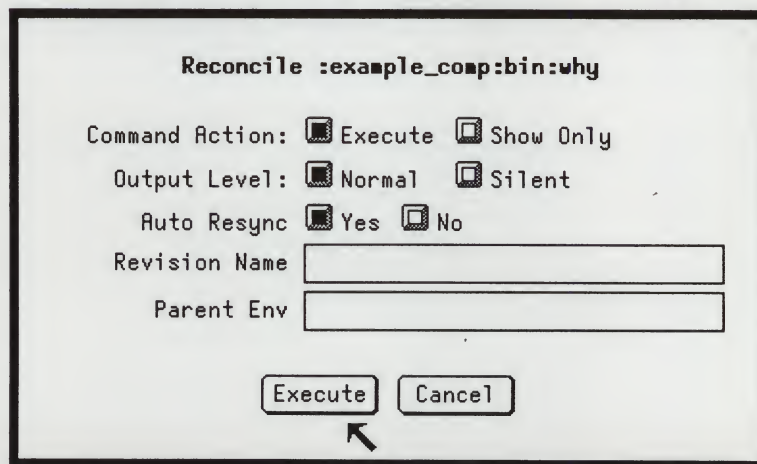
- |            |                                                                                  |
|------------|----------------------------------------------------------------------------------|
| Default    | Simply reconcile without options.                                                |
| Options... | Cause a popup window to appear so the user can specify options to the reconcile. |

Figure 1-109 *Browser: Reconcile =>*



The Options... item on the Reconcile => pull-right menu causes a popup window to appear.

Figure 1-110 *Browser: Reconcile => Options..., Popup Window*



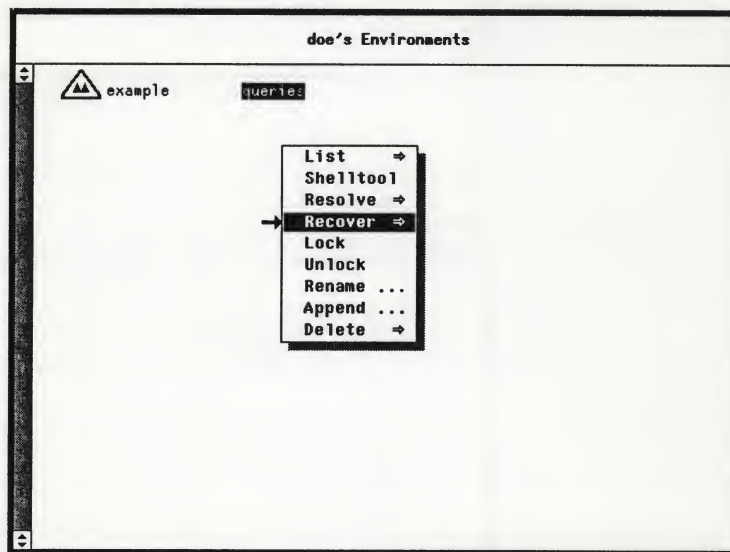
Recover =>

The Recover => operation is an item of either the Environment menu or the Variant menu. Recover => attempts to recover and continue or undo the execution of an unsuccessful command when it is one of the commands listed in the pull-right menu.

Recover => pulls right to a menu with the following items:

|           |                                                                          |
|-----------|--------------------------------------------------------------------------|
| Acquire   | Recover, and if possible continue, an unsuccessful Acquire... operation. |
| Preserve  | Recover, in fact undo, an unsuccessful Preserve... operation.            |
| Reconcile | Recover, in fact undo, an unsuccessful Reconcile => operation.           |
| Resync    | Recover, in fact cancel, an unsuccessful Resync => operation.            |
| Append    | Recover, and if possible continue, an unsuccessful Append... operation.  |

Figure 1-111 Browser: Recover =>



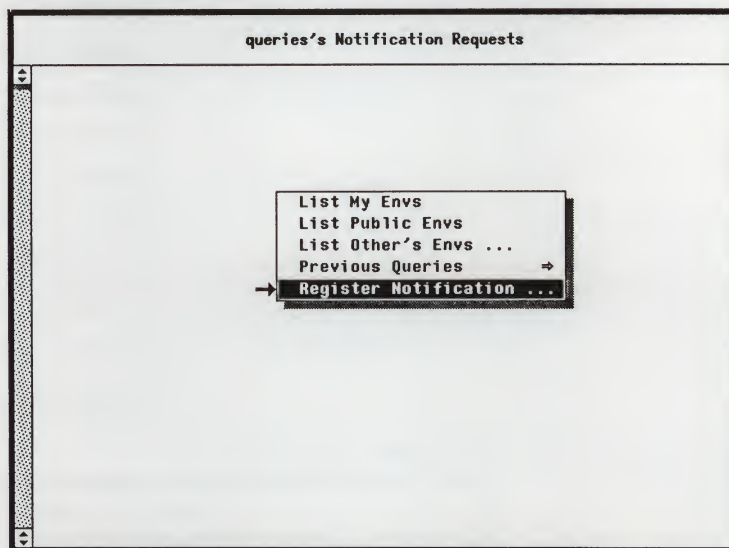
For example, the user might have to execute the Recover => Acquire operation after a machine crash that occurred in the middle of an Acquire... operation.



## Register Notification...

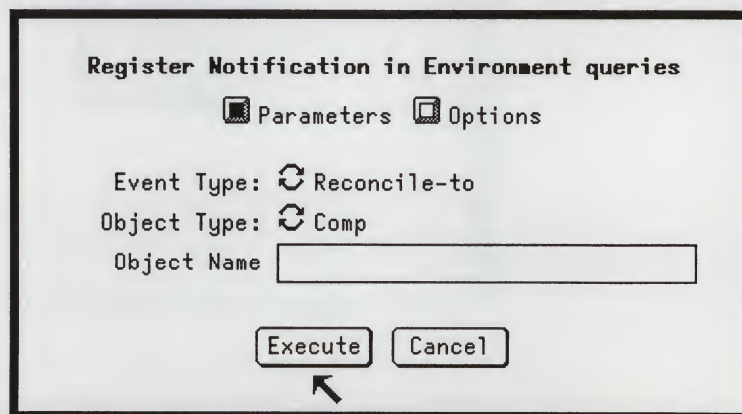
The Register Notification... operation is an item of the Display menu. Register Notification... registers a notification request for execution of a command upon detection of the completion of a certain event.

Figure 1-112 *Browser: Register Notification...*



The user specifies the event type, the object type, and the object name for most events in a dialog box.

Figure 1-113 *Browser: Register Notification..., Dialog Box*



By default, the command executed upon detection of the event is a mail message to the user who registers the request.

The notification facility sorts notification requests by these event types: reconcile-to, acquire-from, resync-from, envlock, and envunlock. Within each of these types, there is a

list of requests, which include: object type and name (optional), recipient (*username@machine*), and command to execute (default is Mail).

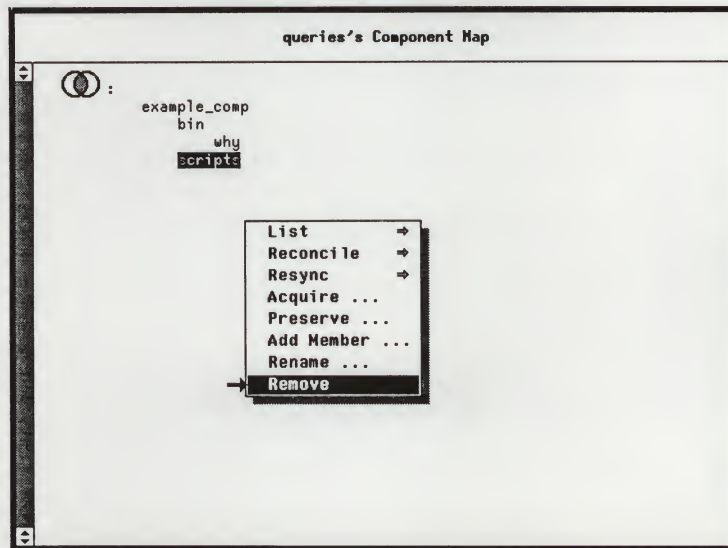
See also the Notifications and Unregister operations.



## Remove

The user can access the Remove operation as an item in either the Component menu, the Target menu, or the Link Database menu. Remove removes the selected object from the current component in the currently activated environment.

Figure 1-114 *Browser: Remove on Component Menu*



Rename...

The Rename... operation is an item on each of these menus:

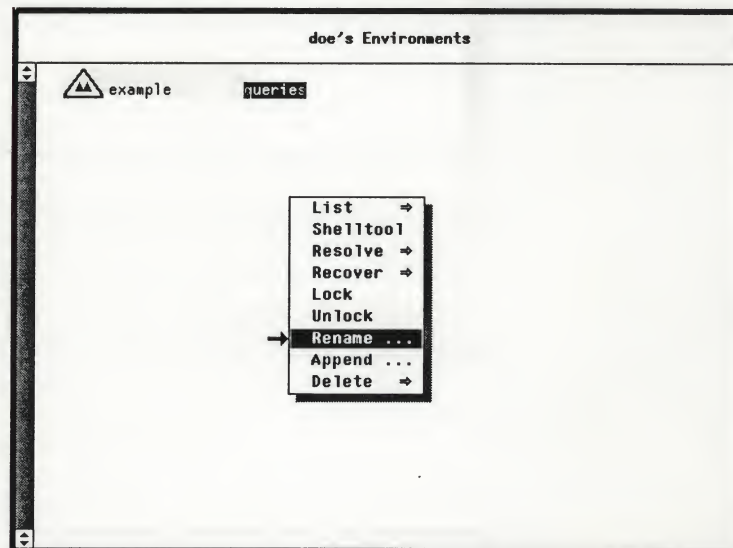
- Environment menu
- Component menu
- File menu
- Link Database menu

Rename... renames the selected environment or object. The user specifies the new name for the environment or object in a dialog box.

Environment Menu

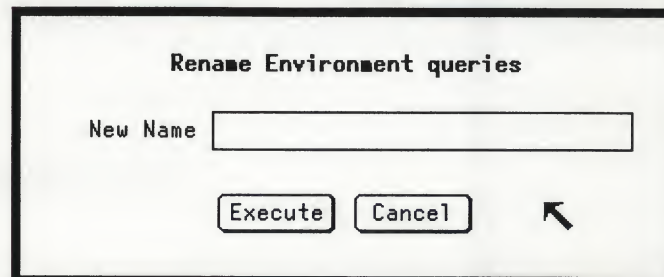
The Rename... item of the Environment menu renames the selected environment.

Figure 1-115 *Browser: Rename... on Environment Menu*



The user specifies the new name for the environment in a dialog box.

Figure 1-116 *Browser: Rename... on Environment Menu, Dialog Box*



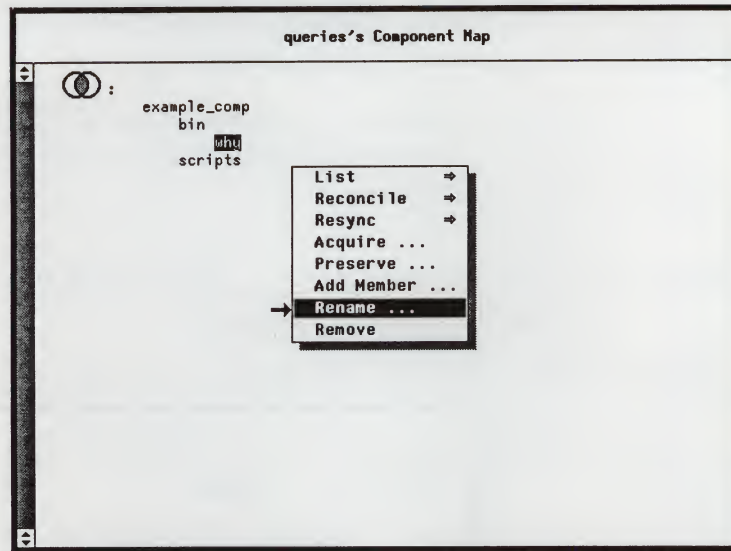
Rename... enables the NSE to track the history of the selected environment.



## Component Menu

The `Rename...` item of the Component menu renames the selected component.

Figure 1-117 *Browser: Rename... on Component Menu*



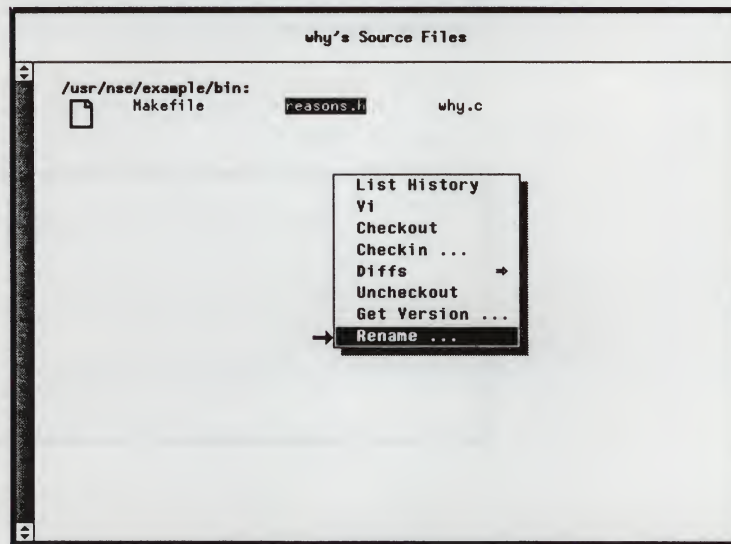
The user specifies the new name for the component in a dialog box.

`Rename...` enables the NSE to track the history of the selected component so that users can later recreate past revisions of the component under the old name of that component.

## File Menu

The `Rename...` item of the File menu renames the selected file.

Figure 1-118 *Browser: Rename... on File Menu*



The user specifies the new filename in a dialog box.

Rename . . . enables the NSE to track the history of the selected file so that users can later recreate past versions of the file under the old name of that file.

The user should use Rename . . . on files instead of the standard SunOS `mv` command so that the NSE can record and track the filenames properly.

When a user renames a file in the NSE, the Browser won't display the renamed files when it lists files until the user changes the name of the file in the Makefiles, components, and `#include` statements that reference the file. This is because the Browser lists files based on dependency information obtained from Makefiles and `#include` statements. Until the user changes the filename in all references to the file, the dependencies still contain the old name.

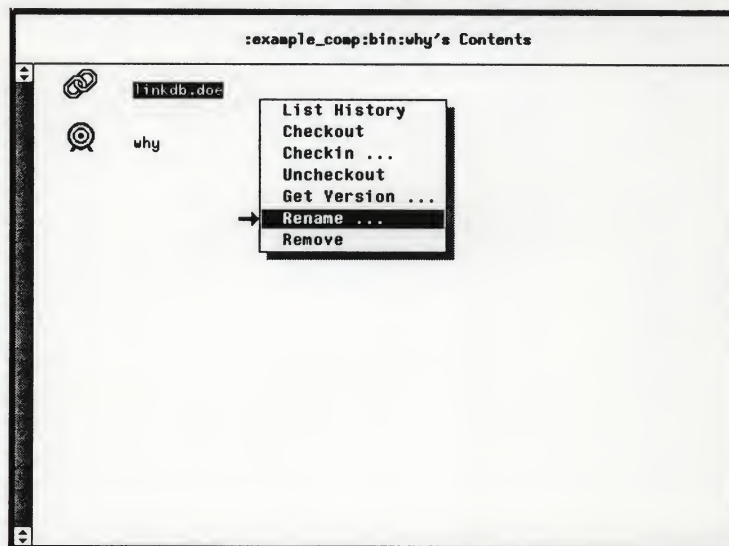
For example, when a user renames an include file `file.h` to `newfile.h`, but the file `prog.c` still includes `file.h`, then listing the source files for target `prog` in the Browser still lists `file.h` rather than `newfile.h`. After the user fixes `prog.c` to include `newfile.h`, the Browser lists `newfile.h` appropriately.

## Link Database Menu

The `Rename . . .` item of the Link Database menu renames the selected link database.

Figure 1-119

*Browser: Rename . . . on Link Database Menu*





The user specifies the new link database name that in a dialog box.

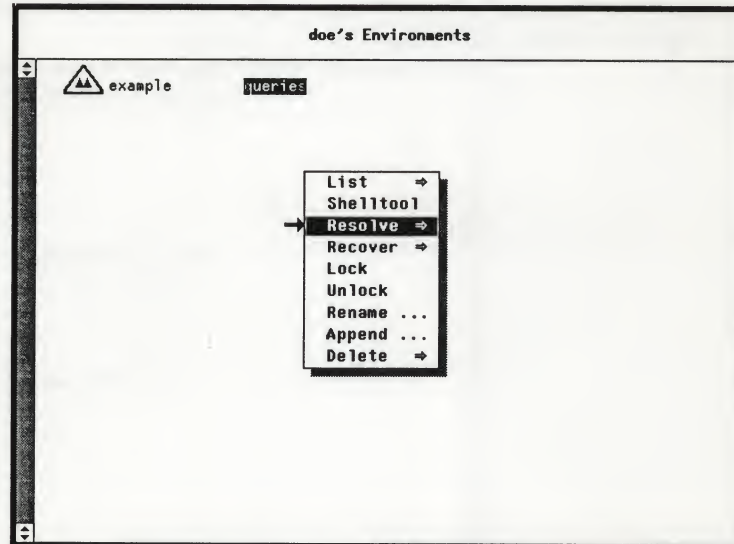
Rename . . . enables the NSE to track the history of the selected link database so that users can later recreate past versions of the link database under the old name of that link database.

Resolve =>

The Resolve => operation is an item of the Environment menu. Resolve => resolves the selected environment with its parent environment. Resolve => pulls right to a menu with the following items:

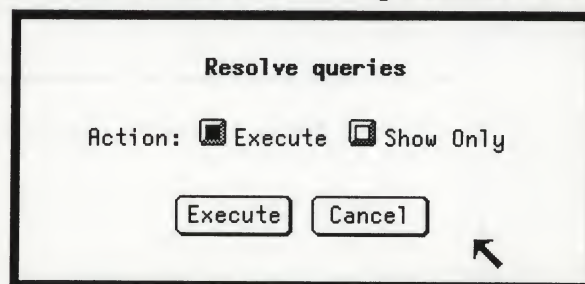
- |            |                                                                  |
|------------|------------------------------------------------------------------|
| Default    | Simply resolves the selected environment.                        |
| Options... | Causes a popup window to appear so the user can specify options. |

Figure 1-120 *Browser: Resolve =>*



The Options... item on the Resolve => pull-right menu causes a popup window to appear.

Figure 1-121 *Browser: Resolve => Options..., Popup Window*





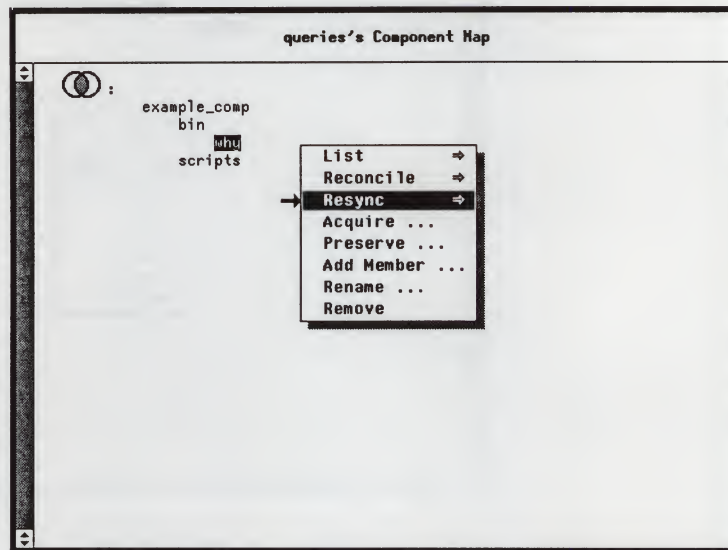
Resync =>

The Resync => operation is an item of the Component menu. Resync => resyncs the selected component in the currently activated environment to its parent environment.

Resync => item of the Component menu pulls right to the following menu items:

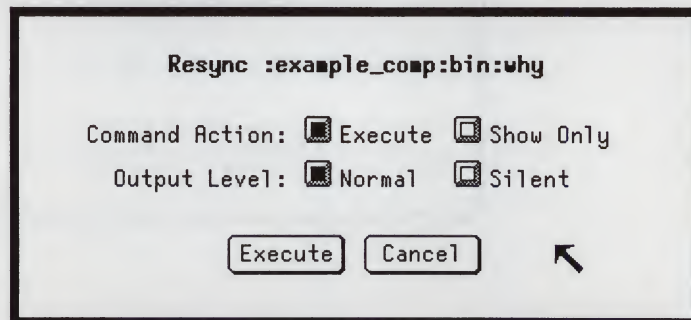
- |            |                                                                               |
|------------|-------------------------------------------------------------------------------|
| Default    | Simply resync without options.                                                |
| Options... | Cause a popup window to appear so the user can specify options to the resync. |

Figure 1-122 *Browser: Resync =>*



The Options... item on the Resync => pull-right menu causes a popup window to appear.

Figure 1-123 *Browser: Resync => Options..., Popup Window*

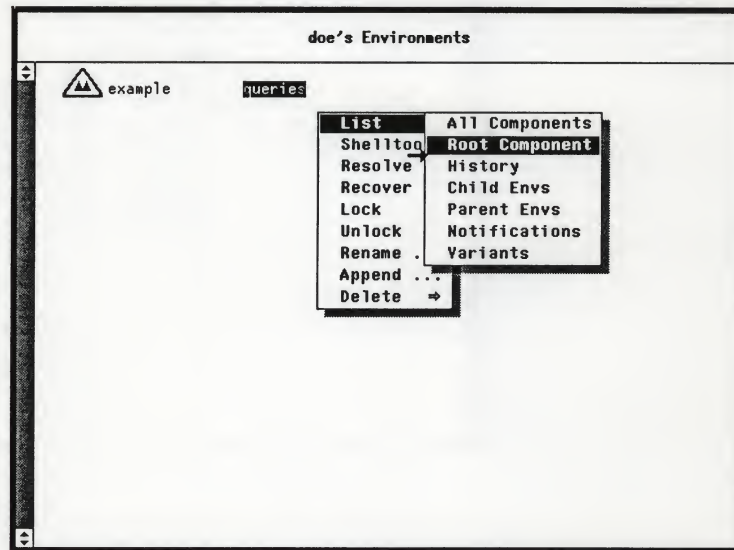


## Root Component

The user can access the Root Component operation from the List => item of either the Environment menu or the Variant menu.

The List => Root Component operation lists the contents of the root component of the selected environment or variant.

Figure 1-124 Browser: Root Component on Environment Menu

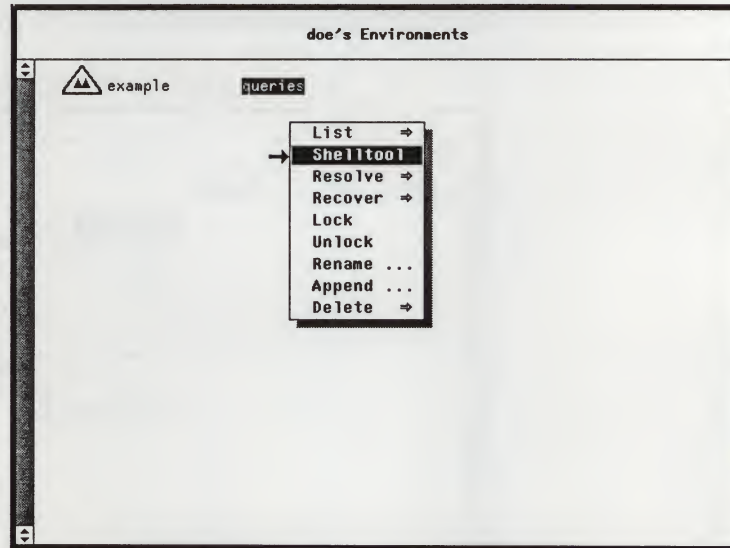




## Shelltool

The Shelltool operation is an item of the Environment menu and the Variant menu. Shelltool invokes a shelltool window activated within the selected environment or variant.

Figure 1-125 *Browser: Shelltool on Environment Menu*

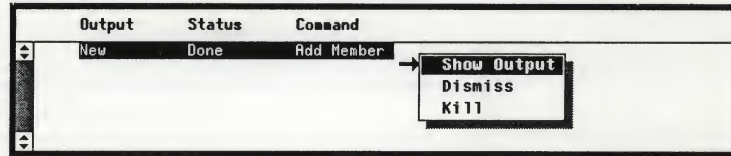


To deactivate the environment and exit the shelltool, type Control-D.

## Show Output

The Show Output operation is an item of the Log Item menu. Show Output causes a popup window to appear which contains the output from the command represented by the selected item in the Log.

Figure 1-126 *Browser: Show Output*





## Textedit

The Textedit operation may be an item of the File menu. The default editor item for the File menu is Vi, but the user may specify Emacs or Textedit as an alternate menu item. For information on how to change the editor item so that it invokes the Textedit operation, see Section 1.8, *Using a Text Editor with the Browser*.

Figure 1-127 Browser: Textedit



Textedit causes a popup window to appear that executes the editor on the selected file.

Figure 1-128 *Browser: Textedit, Popup Window*



```
textedit (linking active) - /usr/nse/example/bin/reasons.h, dir: /usr/nse/example/b
Scratch area ...

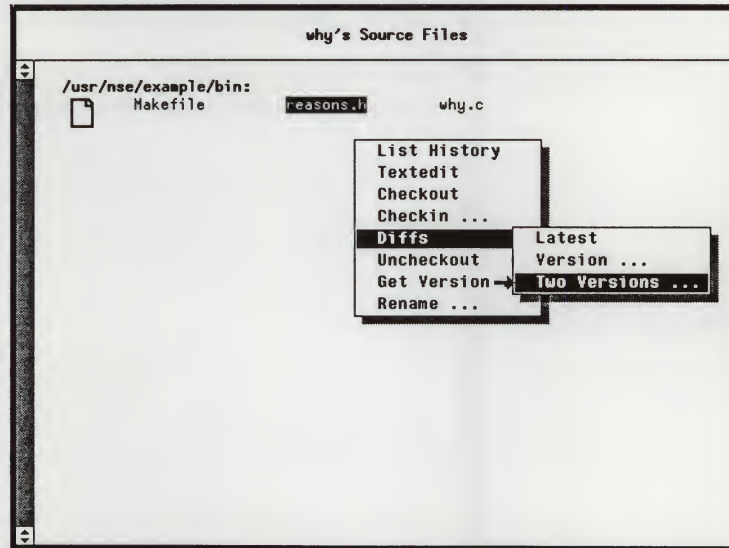
char *reasons[] = {
    "Architectural purity demands nothing less!",
    "Because, that's why.",
    "It's just always been that way.",
    "I dunno",
    "Why not? It has to be some way...",
    "Looks like a feature to me."
};
#define NREASONS ((sizeof reasons)/(sizeof (char *)))
```



## Two Versions...

The user can access the Two Versions... operation through the `Difffs =>` item of the File menu. `Difffs => Two Versions...` compares, using the `filemerge` application, one version of the selected file with another version of the file.

Figure 1-129 *Browser: Two Versions...*



The user specifies the two file version names for the comparison in a dialog box.

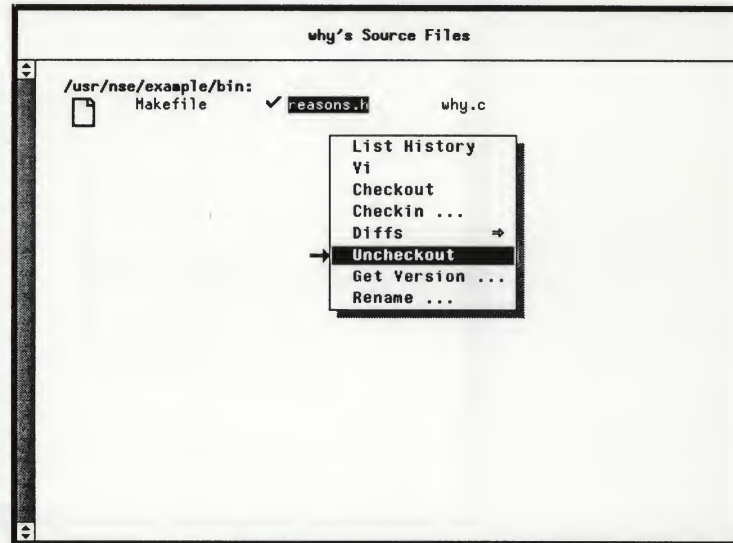
For more information, see the description of the `Difffs =>` operation.

## Uncheckout

The user can access the Uncheckout operation from either the File menu or the Link Database menu.

The Uncheckout item discards the current working version of the selected checked out file or link database. The developer should check out the file or link database again when continuing work on that file or link database.

Figure 1-130 *Browser: Uncheckout on File Menu*



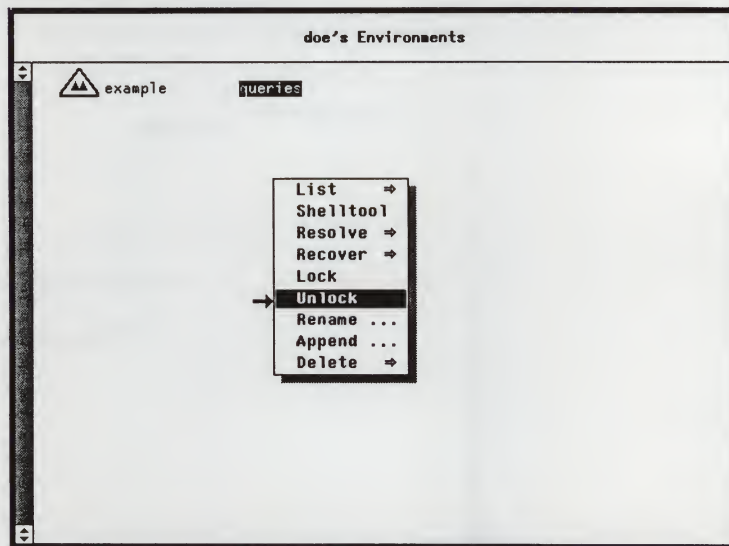
When a user checks out a file or link database from the Version Control System, the Display puts a check mark in front of the name of the file or link database. Checking in, or unchecking out the file or link database, removes the check mark from in front of the name of the file or link database.



## Unlock

The **Unlock** operation is an item of the Environment menu. **Unlock** releases any lock the user made on the selected environment, so that other users can once again make changes to that environment. Users can only release environment locks they created, except for the administrative user **Nse**, who can release locks on any environment.

Figure 1-131 *Browser: Unlock*



See also the **Lock** operation.

## Unregister

The Unregister operation is an item of the Notification menu. Unregister cancels the selected notification request. Only the user who registered the notification request, or the administrative user Nse, can cancel a notification request.

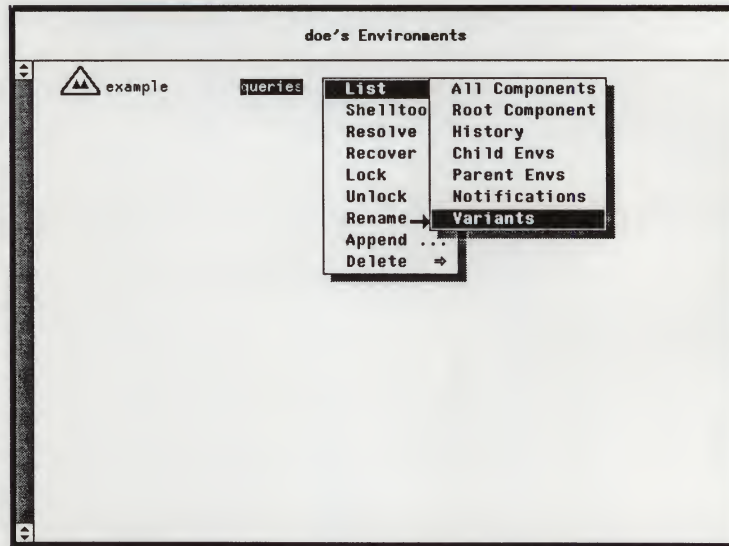
See also the Notifications and Register Notification... operations.



## Variants

The user can access the Variants operation through the List => pull-right menu of the Environment menu. List => Variants lists, in the Display, the names of the variants associated with the selected environment.

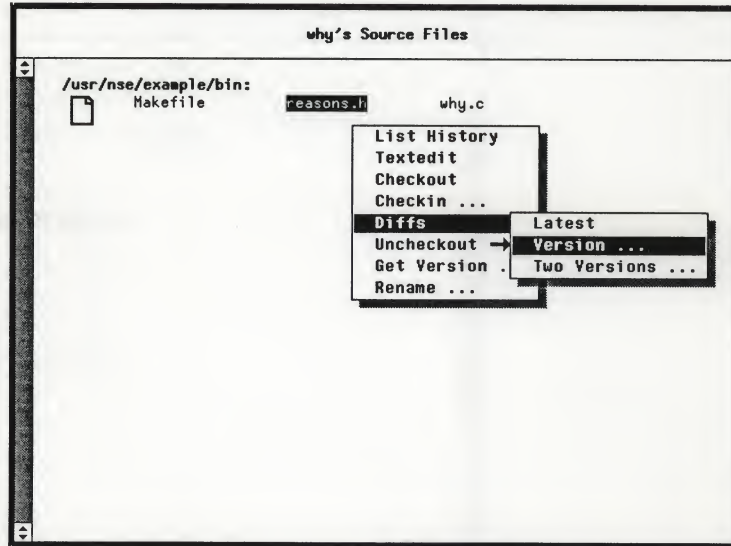
Figure 1-132 *Browser: Variants*



Version...

The user can access the Version... operation through the `Diffs =>` item of the File menu. `Diffs => Version...` compares, using the filemerge application, the current working version of the selected file with another version of the file.

Figure 1-133 *Browser: Version...*



The user specifies the other file version name for the comparison in a dialog box.

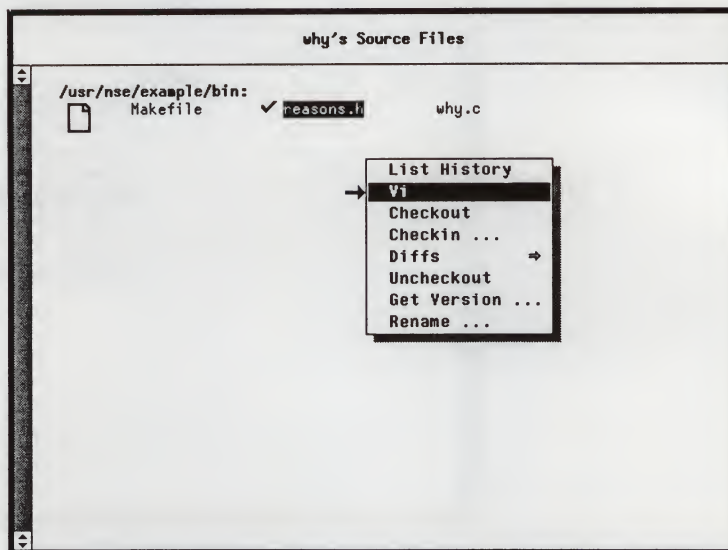
For more information, see the description of the `Diffs =>` operation.



## Vi

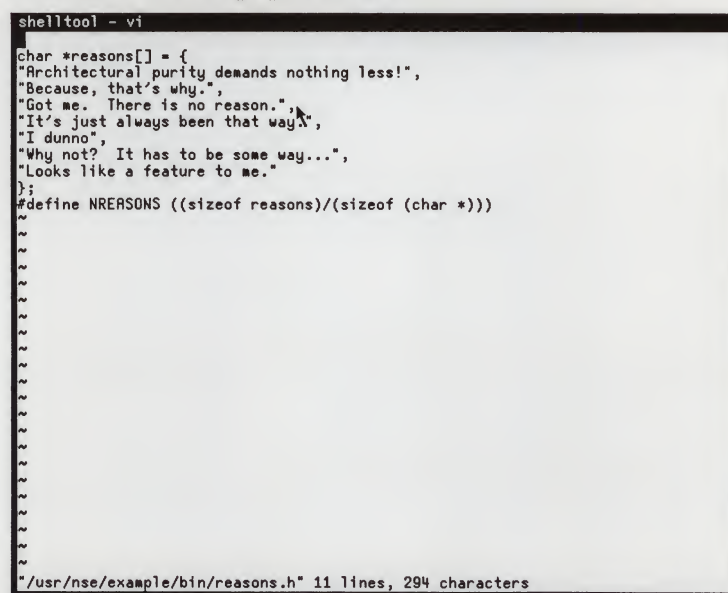
The `Vi` operation is the default editor item for the File menu, but the user may specify `Emacs` or `Textedit` as an alternate menu item. For information on how to change the editor item so that it invokes the `Emacs` or `Textedit` operation, see Section 1.8, *Using a Text Editor with the Browser*.

Figure 1-134 Browser: Vi



`Vi` causes a popup window to appear that executes the editor on the selected file.

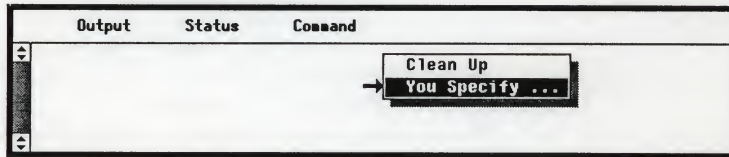
Figure 1-135 Browser: Vi, Popup Window



## You Specify...

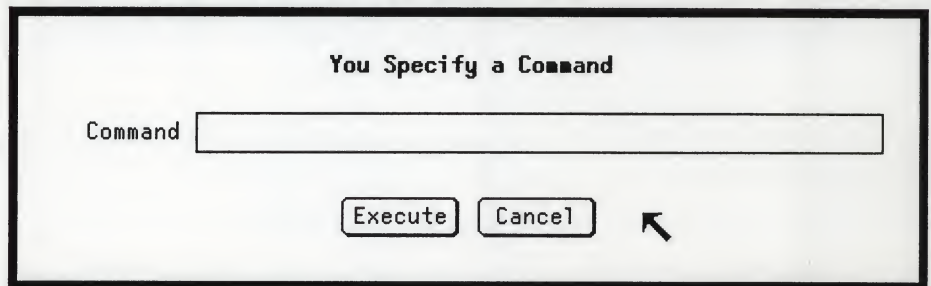
The You Specify... operation is an item of the Log menu. You Specify... permits the user to execute a command of the user's choice in a popup window.

Figure 1-136 *Browser: You Specify...*



The user specifies this command in a popup window.

Figure 1-137 *Browser: You Specify..., Popup Window*





## 1.8 Using A Text Editor with the Browser

The Browser permits you to choose the text editor you want to use. The default Browser file editor is `vi`. Editors the Browser supports include: `vi`, `textedit`, and UniPress Emacs. The user invokes the editor on a selected file in the Display by choosing the editor item of the File menu.

Table 1-2 *Browser: File Editor Operations*

| <i>Browser Operation</i> | <i>Editor Operation Invokes</i> |
|--------------------------|---------------------------------|
| Vi                       | vi                              |
| Textedit                 | textedit                        |
| Emacs                    | UniPress Emacs, or emacs        |

The user can configure the Browser so that the Browser invokes the file editor the user desires. To use `vi`, no configuration is necessary.

To use `textedit`, the user can make a symbolic link between a file called `File.editor.incl` in the user's home directory and the installed file:

```
/usr/nse/lib/menus/File.textedit.incl
```

Were example user `doe` on machine `tutorial` with a home directory of `/usr/doe` to configure her Browser to use `textedit`, she would construct a symbolic link by typing the following command to a SunOS shell prompt:

Figure 1-138 *Browser: File Editor, Symbolic Link*

```
tutorial% ln -s /usr/nse/lib/menus/File.textedit.incl File.editor.incl
tutorial%
```

Then, the Browser editor item on the File menu would appear as `Textedit`, and would invoke the `textedit` file editor on the selected file in the Display.

To use UniPress Emacs, the user must first make sure that the system software includes UniPress Emacs. When the user can't access Emacs outside the Browser, it is unlikely the user will be able to configure the Browser to use Emacs. Then, just as with the configuration for `textedit`, the user can make a symbolic link between a file called `File.editor.incl` in the user's home directory and the installed file:

```
/usr/nse/lib/menus/File.emacs.incl
```

For more information about the `ln(1)` command for making symbolic links, see its Man Page, online or in the *Commands Reference Manual*.





---

## Resolving Objects

|                                   |     |
|-----------------------------------|-----|
| Resolving Objects.....            | 131 |
| 2.1    Resolve.....               | 131 |
| 2.2    Merging With Windows ..... | 132 |
| 2.3    Auto Merging.....          | 132 |
| 2.4    A Sample Resolve.....      | 133 |
| fileresolve.....                  | 133 |
| dbresolve .....                   | 135 |







---

## Resolving Objects

This chapter introduces the NSE facilities for resolving changes made to a development object hierarchy in a child environment with changes made to the corresponding object hierarchy in the parent environment.

One of the primary functions of the NSE is to detect when changes have occurred in files or in other objects that are in separate environments. When the user reconciles a child environment with a parent environment, the `resolve` program reports any conflicts between objects that the user must merge.

The NSE provides two window-based facilities for resolving differences between objects. The facility `fileresolve` handles the more common case of files, while `dbresolve` takes care of other objects whose changes may be more appropriately viewed as a series of database updates. Both programs present practically the same interface to the user; they differ primarily in how completely they display the object the user is merging. `fileresolve` shows each file in its entirety, while `dbresolve` shows only those “records” that users have changed in each object.

### 2.1 Resolve

Use the “show-only” option of `resolve` from `nsebrowse`, or `resolve -n` on the command line if you wish to see a summary of the objects in your environment requiring merging. Invoking `resolve` starts `fileresolve` and `dbresolve` as necessary. `fileresolve` steps through each file needing attention, and then `dbresolve` starts on component descriptions, link databases, and any other objects that need merging.

For a more specific description of each of these programs and an explanation of how to merge objects without a bit-mapped display see `resolve`, `fileresolve`, and `dbresolve` in Chapter 4,



*Command-Line Interface.* The remainder of this chapter describes these commands in general terms, assuming the user is working on a bitmapped display.

## 2.2 Merging With Windows

Both `fileresolve` and `dbresolve` present side-by-side windows showing the object in your environment on the left, and the corresponding object from the parent environment on the right. By comparing the differences in both environments and selecting actions from a command panel, the user composes the merged version of the object in the lower left window. Once the user has resolved all of the differences, pressing `Commit` saves the changes and records the merge.

These programs mark lines which have changes requiring attention with glyphs, small symbols indicating addition (+), deletion (-), or change (|). At any given time, the program highlights one set of glyphs in reverse video, representing the “current” difference. Buttons in the command panel on the lower right control movement through the differences, and the action to take for each. When the user takes an action, the program considers the difference resolved. The program inserts the appropriate lines into the result window below and hollows out the glyphs as a reminder that the user has taken care of that item.

The NSE keeps track of the parallel development of each object, and therefore can identify a “common ancestor” of the objects being merged. The program marks the differences between this ancestor, which is not displayed, and the two objects in question with glyphs in the left and right windows. For example, a column of plus signs on the left indicates that a user added the lines next to the plus signs to the ancestor. In particular, a vertical bar indicates that the line thus marked differs from the ancestor, not necessarily the object on the right.

## 2.3 Auto Merging

With this knowledge of common ancestry, `fileresolve` and `dbresolve` often automatically merge many of the changes. In many cases, the program marks most or all of the glyphs as resolved when the program first loads the objects. The algorithm is different for files and databases, but the program only applies changes when there is no conflict with the object on the other side. The user can disable this marking technique using the property sheet if desired. Both programs permit the user to examine those

## 2.4 A Sample Resolve

changes which have already been automerged to assure that the correct choices have been made.

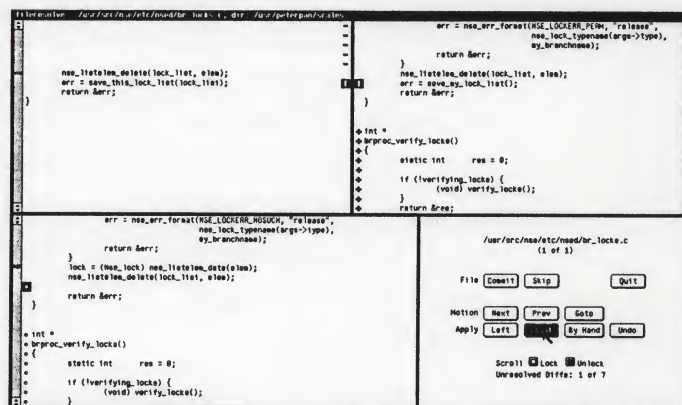
A typical resolve situation is often as follows: both you and someone else have changed some of the same files and also added subcomponents to the same component. Assuming that the other person has already reconciled with your common parent environment, you now must resolve these conflicts before your reconcile can succeed. Either you just tried to reconcile and were told that objects had changed, or you decided to resync this component to get these latest changes. Either way, running resolve is the next step.

From `nsebrowse`, the user invokes a resolve command by selecting an environment and choosing **Resolve** from the Environment menu. From the command line, typing `resolve` while activated in the environment has the same effect.

`fileresolve`

`Resolve` takes a moment to start `fileresolve`. Then the program loads the first file requiring merging. By default, **Auto Merge** is set on, so when the windows paint completely, many of the glyphs are already hollowed out. The **Unresolved Differences** message in the lower right also indicates that only a few of the differences require your attention.

Figure 2-1 `fileresolve`



`fileresolve` scrolls the windows so that the next unresolved difference is in view, and highlights the glyphs associated with that difference. Pressing **Next** brings up the next unresolved difference. When you wish to examine the choices made by the automerge algorithm, use the **Next resolved** option of the



Next button. This is located on the menu behind the button, and accessible by holding down the Control and Shift keys together while pressing the button. As in mailtool(1), most of the buttons have menus with accelerators.

For each difference, choose either Left or Right to apply the changes in the corresponding windows above. Pressing Left is equivalent to saying "take my changes," while pressing Right accepts those that were made in the parent environment. In some cases, it may be necessary to resolve the difference "by hand." In this case, use the editing features of the bottom text subwindow and then press the By Hand button to inform fileresolve that you have resolved this difference. For more information on editing, see the textedit(1) Man Page, online or in the *Commands Reference Manual*.

By default, fileresolve automatically advances to the next unresolved difference after taking one of these actions. This is another characteristic which the user may change with the property sheet.

When the user reduces the number of unresolved differences to zero, the merge is complete and the user may commit the merge. fileresolve prompts the user for confirmation when the user attempts to commit the merged with some unresolved differences remaining. Pressing Commit will save the merged changes, check the file in using the Version Control System, and move on to the next file.

Occasionally, the user will decide simply to accept the other version of the file. This happens frequently enough that Accept Parent is one of the options behind the Commit button.

At any time, the user may skip the current file by clicking on the Skip button, and proceed to the next file. By using the menu behind the Skip button, the user can go to any other file still in need of merging.

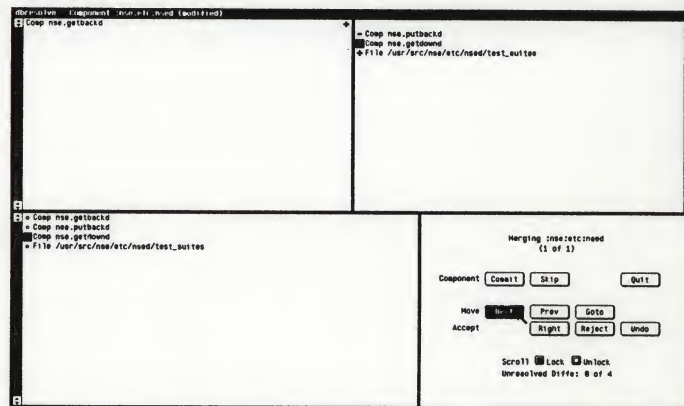
In the middle of a particularly long resolve, the user may Quit from fileresolve and the next resolve will pick up where the user left off.

When the user has merged the last file requiring attention, fileresolve exits and resolve starts dbresolve to handle the component merge.

## dbresolve

With dbresolve, as with fileresolve, the user's changes appear on the left, the changes in the parent on the right. Unlike fileresolve, however, dbresolve displays only those database "records" that have changed. The user can't edit the bottom window directly and the choices are simply to Accept, Left, Right, or Reject each record. As the user addresses each difference, dbresolve updates the bottom window to reflect the changes the user will eventually commit.

Figure 2-2 dbresolve



dbresolve automerges before the windows load, and the user may find that there is not much to do by hand. The user may use the Next button as described above to examine the changes.

When the user completes the merge, the user presses the Commit button to apply the changes to the component in the user's environment. When there is another component that requires merging, dbresolve loads the next component; otherwise dbresolve exits.

When there are other conflicts, for example, between Link Service databases, dbresolve starts again following the component merges to handle the conflicts.

After resolving all of the differences, the "show-only" option of resolve in nsebrowse, or the `resolve -n` command provides the message:

No objects require merging.





## Links and the Link Service

|                                                    |     |
|----------------------------------------------------|-----|
| Links and the Link Service.....                    | 139 |
| 3.1    Introduction.....                           | 139 |
| 3.2    Link Databases.....                         | 140 |
| 3.3    The Link Service Operating Environment..... | 140 |
| 3.4    User Interface.....                         | 141 |
| Starting up a Link Service Window.....             | 141 |
| Link Service Window Header Information.....        | 141 |
| Link Service Popup Menus.....                      | 142 |
| 3.5    Link Service Operations .....               | 143 |
| Cancel .....                                       | 143 |
| Confirm All.....                                   | 144 |
| Create .....                                       | 144 |
| Database.....                                      | 144 |
| Delete .....                                       | 145 |
| Display .....                                      | 145 |
| End Link .....                                     | 146 |
| Set .....                                          | 147 |
| Start Link .....                                   | 147 |







---

## Links and the Link Service

### 3.1 Introduction

The NSE Link Service permits users to connect related entities in different applications. A *link* is the association that the user specifies between two entities using the Link Service. For example, a user could connect one entity – a source code function listed in a `textedit` window – to another entity – the documentation associated with that function in a document preparation package. The user could also specify a link between the function listed in a `textedit` window and that function's design as displayed in a design tool. Likewise, the user could link the design to the documentation. Whenever the user works with integrated tools that access the entities involved in the software development process, the Link Service permits the user to specify and track automatically any other entities of interest.

Using the Link Service, the user can make associations between entities in different applications that persist over time. This allows users to document relations between applications that access similar entities. Developers and others who inherit a project can track and maintain these links, providing valuable information about the relation between previously unrelated applications. Thus, the user can achieve greater integration between the applications that address and provide support for the entire software life cycle.

Because the NSE provides a platform with which software developers can integrate their own packages, the Link Service can link together application entities within many products from a variety of vendors.

This chapter describes:

- Link Databases
- The Link Service Operating Environment
- Link Service User Interface
- Link Service Operations



The Link Service includes an interactive interface used to link together entities from various applications, as well as to delete, confirm, and display those links. The window-based interactive interface runs only on bit-mapped screens. It includes a menu of commands for linking, deleting, canceling a halfway completed link, displaying existing linked entities, and operating on link databases.

### **3.2 Link Databases**

Link databases store the links specified by Link Service users and provide controls that prevent problems with concurrent database access.

At present, the Link Service provides no system security beyond that of SunOS permissions. The Link Service creates link database files with permissions set so that any user can read or write them. However, a user can change these permissions so that, for example, only the owner of a link database has write permission for that link database.

### **3.3 The Link Service Operating Environment**

Whenever the user starts a Link Service within an activated NSE environment, the Link Service binds itself to that environment and any applications associated with that environment.

Whenever the user invokes an application from within an activated environment, that application associates itself with that activated environment. However, some applications, like the NSE Browser, can change their association from one environment to another.

The user must have a Link Service running for each environment on each machine where the user wants to create or display links. When a user activates more than one environment at a time, and starts a Link Service within one of these activated environments, a given Link Service can only communicate with the applications associated with that activated environment.

Users can only create or display links between applications associated with the same activated environment.

There may be at most one Link Service active simultaneously for each NSE environment.

When the user starts a Link Service outside an activated environment, the user can't use that Link Service to create or follow links within any NSE environments.

A user can switch the Link Service between link databases available within a particular activated environment. When the user

wants to obtain a link database from another environment, the user must either add the link database using the `nsecomp add` command, or acquire a component that contains the link database using the `acquire` command.

The relationship between databases, projects, and users is up to the users; in other words, users may choose to have one link database for each software project, for each user, or for each software project and target architecture combination.

### 3.4 User Interface

This section describes the user interface for the Link Service.

#### Starting up a Link Service Window

To start up a Link Service window, type `linksvc` to the SunOS shell prompt, followed by the ampersand character (`&`), to run the program in the background:

Figure 3-1 *Starting Up a Link Service Window*

```
tutorial% linksvc &  
tutorial%
```

#### Link Service Window Header Information

When the Link Service icon is open, the Link Service displays context information in the window header of the Link Service window. The window header lists the command name, followed by an NSE environment name, if any, and by the currently active database, if any, all of these elements separated by colons.

Figure 3-2 *Link Service Window Header*



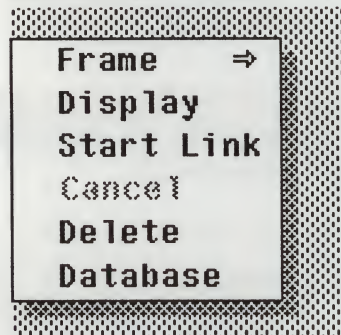


## Link Service Popup Menus

After you start up the Link Service, a window appears, with a panel that displays a group of graphical buttons. Descriptions of these Link Service operations appear in Section 3.5, *Link Service Operations*.

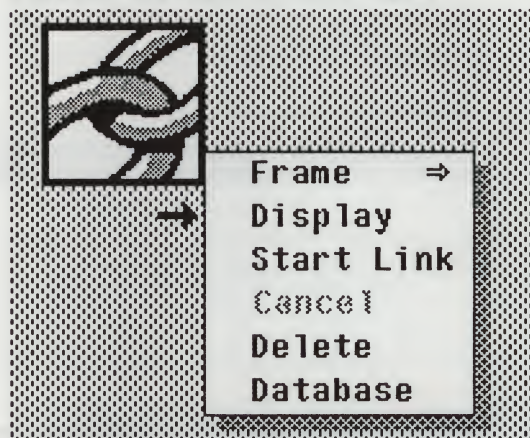
When you press and hold down the right mouse button while on the border, which includes the window header, of a Link Service window, a pull-right menu of these commands appears, preceded by the menu item, **Frame**:

Figure 3-3 *Link Service Window: Frame Menu*



This same pull-right menu appears if you press the right-most button after moving the cursor over the Link Service's icon, when the window is closed:

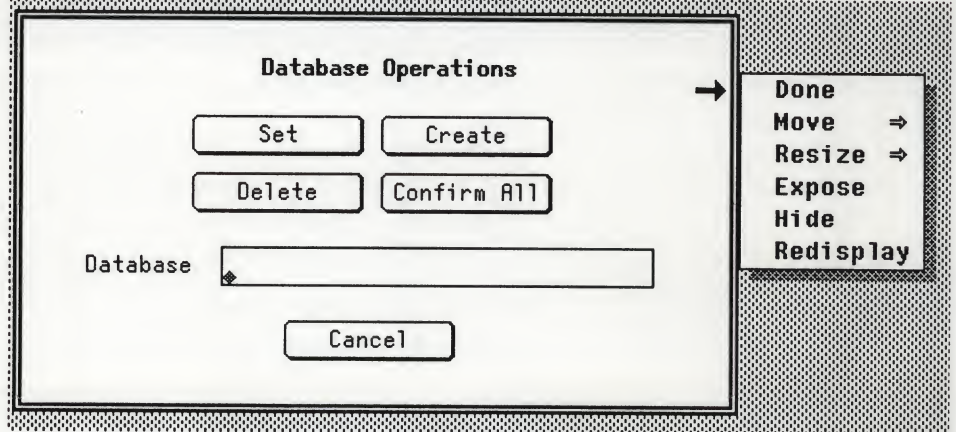
Figure 3-4 *Link Service Icon: Frame Menu*



When you pull right on this **Frame** menu item, you get the standard frame menu with the following menu items: **Close or Open**, **Move**, **Resize**, **Expose**, **Hide**, **Redisplay**, and **Quit**. For more information on the **Frame** menu and the operation of window-based tools, see the *Windows and Window-Based Tools: Beginner's Guide*.

All popup subwindows, except the main window, have a subwindow pull-right menu. To display the pull-right menu, move the cursor into the border of the popup subwindow until the cursor changes to a circle, then hold down the right mouse button.

Figure 3-5 *Link Service Database Popup Window: Pull-Right Menu*



### 3.5 Link Service Operations

This section describes following the Link Service operations in alphabetical order:

- Cancel
- Confirm All
- Create
- Database
- Delete
- Display
- End Link
- Set
- Start Link

All operations may result in the shading of graphical buttons to indicate those buttons that are not operational at that point in the operation. Click-ahead does not work when this is the case.

Cancel

The Cancel graphical button appears on the Link Service window, as well as all of the Link Service popup windows. On the popup windows, clicking the Cancel button simply cancels the current operation.

Just after starting the Link Service, the Cancel graphical button in the Link Service window appears shaded because the user can't specify a Cancel operation until after the user starts another



operation. The Cancel command is available in the Link Service window only when the Link Service has successfully begun the first half of a link. This occurs when the Link Service replaces the Start Link graphical button with the End Link graphical button. At this time, the user can cancel the halfway created link by clicking on the Cancel button. The Cancel command once again becomes inactive, and the Link Service replaces the End Link button with the Start Link button.

Choosing the Start Link graphical button causes the button to appear as End Link so the user can indicate the end of the linking operation. The Cancel button is functional, and not shaded during the linking operation. Choosing the End Link graphical button causes the button to appear as Start Link, and the Cancel operation becomes shaded and no longer functional.

#### Confirm All

The Confirm All operation asks the Link Service to confirm the existence of all of the entities in the database named in the text field. For each entity in the link database, there must be an application currently running that is capable of displaying that entity. For example, when a database has entities from NSE source files to which a user made links while displaying the entities in `textedit`, then there must be a `textedit` running while the user executes the Confirm All operation. When the user has a large database, the confirm operation may take a very long time.

If all of the entities in a database are valid, then Confirm All causes a confirmation window to appear. If there are invalid entities in the database, then Confirm All causes a popup subwindow to appear with a list of all the invalid links, an asterisk ( \* ) listed by each of the entities that are invalid.

The user may view the list and either delete all of the invalid links in the list, or quit the confirm operation, leaving the database intact.

#### Create

The Create command creates a database and sets that database to be the currently active database for the Link Service and applications in that environment. Then, the database popup window disappears. The database must not already exist.

#### Database

The Database button results in another popup menu with the following items: Set, Delete, Create, Confirm All, and

Cancel buttons, and a text field for filling in the name of the database on which these commands should operate.

The currently active database is displayed in the window header of the main window, and is the default setting for the database's popup subwindow text entry box. To operate on this or any other database, the name of the database must be in the text entry box.

To quit the Database dialog box, choose the Done item from the Frame menu of the Database dialog box or the Cancel button on the Database popup subwindow.

## Delete

There are two possible Delete operations:

- Delete a link database
- Delete a link

When invoked from the database popup, the Delete command removes the database named in the text field. The command asks the user for confirmation before actually destroying the database and removing the Database popup window.

When invoked from the main menu, the Delete operation removes the specified link.

Unlike creating a link, deleting a link does not require that the application that can display the other part of the link be running. To delete a link, select the entity in an application whose link in another application is to be deleted. Then select the Delete command. When the selected entity has only one link, that link is destroyed. When the selected entity has links to multiple entities, the Link Service pops up a subwindow, with labels for each of the entities with links to the initial selection.

To delete all links to the entity initially selected, choose the All button. To delete individual entities, toggle those you desire to delete, so that there is a check in the box to the left of each of their labels. When you are ready to delete those links, choose the Execute button. To quit the Delete popup window, choose the Done item from the Frame menu of the Delete dialog box or the Cancel button on the Delete popup subwindow.

## Display

Displaying links allows a user to select an entity in one application and have an entity to which it was previously linked in another application appear on the screen, ready to be manipulated. To



display a link, select the entity in one application whose link in another application is to be displayed. Then select the `Display` graphical button.

When the selected entity has only one link, and when an application that can display the entity associated with that link is running, `Display` traverses the link from one entity to the entity associated with that entity, and displays the associated entity.

When the selected entity has links to multiple entities, the Link Service pops up a subwindow with labels for each of the entities. The user chooses which entities to display by toggling them so there is a check in the box to left of each of their labels.

When there is only one application running that can display a chosen entity, `Display` displays the entity.

When there is more than one application running that can display the entity, another popup subwindow appears with the list of which applications can display the entity. The user chooses one application for displaying the associated entity, and the popup disappears. When a user decides not to display the entity, the user can choose the `Done` item on the subwindow pull-right menu to quit the popup or the `Cancel` button on the `Display` popup window.

`Display` results in another popup menu if there is more than one link for the entity selected. `Display` will also result in another popup menu if there is more than one application registered with that Link Service in the same class for an entity that is to be displayed.

#### End Link

The user selects an entity in an application, and selects the `Start Link` command. This item then becomes `End Link`. The user selects the second entity to be linked to the first, and finally selects the `End Link` command to complete the link.

Choosing the `Start Link` graphical button causes the button to appear as `End Link` so the user can indicate the end of the linking operation, and the `Cancel` operation becomes unshaded and functional during the linking operation. Choosing the `End Link` graphical button causes the button to appear as `Start Link`, and the `Cancel` operation becomes shaded and no longer functional.

**Set**

This command makes this the currently active database and removes the Database dialog box. The database must already exist, and not already be the currently active database.

**Start Link**

The user selects an entity in an application, and selects the **Start Link** command. This item then becomes **End Link**. The user selects the second entity to be linked to the first, and finally selects the **End Link** command to complete the link.

Choosing the **Start Link** graphical button causes the button to appear as **End Link** so the user can indicate the end of the linking operation, and the **Cancel** operation becomes unshaded and functional during the linking operation. Choosing the **End Link** graphical button causes the button to appear as **Start Link**, and the **Cancel** operation becomes shaded and no longer functional.





## Command-Line Interface

|                                                      |     |
|------------------------------------------------------|-----|
| Command-Line Interface.....                          | 151 |
| 4.1  Command Prompt for Activated Environments ..... | 151 |
| 4.2  Command-Line Interface Example.....             | 152 |
| Setting Up a Command-Line Interface Example.....     | 152 |
| Running a Command-Line Interface Example .....       | 153 |
| 4.3  Command Summaries.....                          | 163 |
| acquire.....                                         | 164 |
| activate .....                                       | 168 |
| append .....                                         | 170 |
| bootstrap.....                                       | 173 |
| dbresolve .....                                      | 175 |
| describe_comps.....                                  | 179 |
| fileresolve.....                                     | 181 |
| linksvc.....                                         | 186 |
| lvcs .....                                           | 187 |
| merged.....                                          | 189 |
| nsebrowse.....                                       | 193 |
| nsecomp .....                                        | 195 |
| nsecontrolpt.....                                    | 199 |
| nseenv .....                                         | 201 |
| nseexecset .....                                     | 213 |





---

|                 |     |
|-----------------|-----|
| nsegc .....     | 218 |
| nseivp .....    | 220 |
| nsemark .....   | 221 |
| nsenotify ..... | 222 |
| nsemlink .....  | 226 |
| nsevar .....    | 229 |
| preserve .....  | 233 |
| reconcile ..... | 236 |
| resolve .....   | 241 |
| resync .....    | 243 |
| vcs .....       | 246 |

## Command-Line Interface

The NSE command-line interface permits users on terminals without bit-mapped screens, and users who write script files to access the NSE. In most cases, users find the window- and mouse-based NSE Browser the most effective way to use the NSE. For more information about the NSE Browser, see Chapter 1, *The Browser*.

This chapter provides a description of how to make your command prompt indicate an activated environment and provides a detailed syntax and use summary for each command.

### 4.1 Command Prompt for Activated Environments

When you activate an environment, you may find it useful for your command prompt to list the name of the activated environment, so you can keep track of which environments you've activated in which windows.

Here is an example expression you can insert into the `.cshrc` file in your home directory to cause the command prompt to track the name of the activated environment:

Figure 4-1 *Command Prompt Insert for .cshrc File*

```
if ($?prompt) then
  if ( $?NSE_ENV ) then
    set prompt = "`/bin/hostname` (${NSE_ENV})% "
  endif
endif
```

You can insert this expression using the text editor of your choice. Insert the expression at the beginning of the `.cshrc` file **after** any unconditional "set prompt =" expressions. Inserting the expression as described reduces the chance of interfering destructively with other portions of the file.



For more information on altering your command prompt, see the *Setting Up Your UNIX Environment: Beginner's Guide*.

## 4.2 Command-Line Interface Example

This command-line interface example will help you to understand the overall process of using the NSE command-line interface. The most frequent sequence of NSE commands is an acquire/reconcile/resolve cycle.

### Setting Up a Command-Line Interface Example

To set up an acquire/reconcile/resolve example, you can run an example script distributed with the NSE.

The example script creates an environment called `example`, so check to make sure that you do not already have an environment called `example` or an environment called `queries` by using the `nseenv list` command.

Figure 4-2 *Checking Your Environment List*

```
tutorial% nseenv list
tutorial%
```

When the `nseenv list` command doesn't return anything, as above, you don't own any environments. Should the `nseenv list` command return a list of environment names that includes the environment `example`, the environment `queries`, or both, make sure to rename or delete the environment or environments using one of the `nseenv` commands before continuing with the example setup procedure.

**Note:** The script `create_example` adjusts to the machine architecture which, in this listing, is the Sun-3 architecture.



To set up the example, invoke the example script  
`create_example`:

Figure 4-3 *Setting Up a Command-Line Interface Example*

```
tutorial% create_example
/bin/arch
nseenv create -v sun3 example /usr/nse/example
bootstrap example /usr/nse/example/example.cmp
Adding components
Components added

Verifying that all objects are stable:
Checking:  Comp :example_comp
...
Checking:  Comp :example_comp:scripts

Now preserving objects:
Preserve:  Dir /usr/nse/example
...
Preserve:  Comp :example_comp
Acquiring:
Comp :example_comp, revision 'bootstrap', number 1, created 11/05/87 17:24:48 by
doe.
Acquire new:  Dir /usr/nse/example
...
Acquire new:  Comp :example_comp
cd /usr/nse/example/bin
activate example make
cc -g -o why why.c
echo Initial base
activate example preserve -bp
Verifying that all objects are stable:
Checking:  Comp :
...
Checking:  Comp :example_comp:scripts

Now preserving objects:
Preserve:  Dir /usr/nse/example
...
Preserve:  Comp :

Now saving files:
tutorial%
```

### Running a Command-Line Interface Example

Running this command-line interface example consists of the following steps:

- Acquire a component into a child environment.



- Activate the child environment.
- Make changes to a file and update the Version Control System.
- Build and test the changed component.
- Make a revision of the changed component.
- Reconcile the component in the child environment with the corresponding component in the parent environment.
- Resolve the changes made in the component of the child environment with the corresponding component in the parent environment.

The first step is to acquire a new component, `:example_comp` in the example, from environment `example` into a newly created child environment, in this case `queries`. Then, activate the child environment `queries` so you can access the new component in the child environment:

Figure 4-4 *Acquire Component and Activate Child Environment*

```
tutorial% acquire -c queries example :example_comp
Acquiring:
Comp :example_comp, revision 'base_rev', number 1, created 11/05/87 21:47:48 by
doe.
Acquire new: Dir /usr/nse/example
...
Acquire new: Comp :example_comp
tutorial% activate queries
tutorial(queries)%
```

When you activate the environment `queries`, the command prompt won't change to indicate the name of the activated environment unless you modify the `.cshrc` file in your home directory. For information on how to change your `.cshrc` file, see Section 4.2, *Command Prompt for Activated Environments*.

Now, check out the `reasons.h` file from the Version Control System:

Figure 4-5 *Checking Out a File from the Version Control System*

```
tutorial(queries)% cd /usr/nse/example/bin
tutorial(queries)% vcs checkout reasons.h
1.1
new version 1.2
10 lines
tutorial(queries)%
```



Using your favorite text editor, such as `vi`, add the following line to the file `reasons.h`, as the first in the list of reasons "why":

Figure 4-6 *Line to Add to reasons.h File*

```
"Because I told you so.",
```

List `reasons.h` to make sure the file looks like this:

Figure 4-7 *Listing the reasons.h File*

```
tutorial(queries)% more reasons.h

char *reasons[] = {
  "Because I told you so.",
  "Architectural purity demands nothing less!",
  "Because, that's why.",
  "It's just always been that way.",
  "I dunno",
  "Why not? It has to be some way...",
  "Looks like a feature to me."
};
#define NREASONS ((sizeof reasons)/(sizeof (char *)))
tutorial(queries)%
```

Check in the file `reasons.h` to the Version Control System:

Figure 4-8 *Checking In a File to the Version Control System*

```
tutorial(queries)% vcs checkin reasons.h
Comment?
> Added another reason why to reasons.h header file
>
<
1.2
1 inserted
0 deleted
10 unchanged
tutorial(queries)%
```

Make sure that the changed component builds:

Figure 4-9 *Building the Changed Component*

```
tutorial(queries)% make
cc -g -o why why.c
tutorial(queries)%
```



**Note:** The `why` program may return a variety of responses.

At this point, run any tests you believe can confirm the results of the build. For example, invoke the program `why` to find out why:

Figure 4-10 *Testing the Changed Component*

```
tutorial(queries)% why
Why not? It has to be some way...
tutorial(queries)%
```

Finally, attempt a reconcile of the changed component from the child environment into the parent environment, in this example a reconcile component `queries` with component example:

Figure 4-11 *Attempt reconcile of Changed Component: No Changes in Parent*

```
tutorial(queries)% reconcile :example_comp
Reconciling with example:
Current Comp :example_comp

Determining what objects to reconcile:
Checking:  Comp :example_comp
...
Checking:  Comp :example_comp:scripts
Reconcile update:  File /usr/nse/example/bin/reasons.h
Reconcile update:  File /usr/nse/example/bin/why
Preserve:  Targ why%/usr/nse/example/bin/Makefile
Preserve:  Comp why
Preserve:  Comp bin
Preserve:  Comp :example_comp

Now saving files:
tutorial(queries)%
```

When no one has changed the component in the parent environment, the `reconcile` works immediately, updating the component in the parent environment automatically with the changes made to the component in the child environment.



However, when someone has made some changes to the component in the parent environment, the procedure is a bit more complex. To simulate a change in the parent environment, invoke the `parallel_update` script:

Figure 4-12 *Simulating a Change in the Parent Environment*

```
tutorial(queries)% parallel_update
cd /usr/nse/example/bin
vcs co reasons.h
1.2
new version 1.3
11 lines
sed -e s/bug/feature/ reasons.h
mv new_reasons.h reasons.h
make
cc -g -o why why.c
echo Changed bug to feature
vcs ci reasons.h
Comment?
> > <
1.3
0 inserted
0 deleted
11 unchanged
preserve -cParallel update :example_comp

Verifying that all objects are stable:
Checking: Comp :
...
Checking: Comp :example_comp:scripts

Now preserving objects:
Preserve: File /usr/nse/example/bin/why
...
Preserve: Comp :

Now saving files:
tutorial(queries)%
```



Once again, check out the file `reasons.h` from the Version Control System:

Figure 4-13 *Checking Out a File from the Version Control System*

```
tutorial(queries)% cd /usr/nse/example/bin
tutorial(queries)% vcs checkout reasons.h
1.2
new version 1.3
11 lines
tutorial(queries)%
```

Using your favorite text editor, such as `vi`, add the following line to the file `reasons.h`, as the first in the list of reasons “why”:

Figure 4-14 *Line to Add to reasons.h File*

```
"No reason. It's pointless.",
```

List `reasons.h` to make sure the file looks like this:

Figure 4-15 *Listing the reasons.h File*

```
tutorial(queries)% more reasons.h

char *reasons[] = {
    "No reason. It's pointless.",
    "Because I told you so.",
    "Architectural purity demands nothing less!",
    "Because, that's why.",
    "It's just always been that way.",
    "I dunno",
    "Why not? It has to be some way...",
    "Looks like a feature to me."
};
#define NREASONS ((sizeof reasons)/(sizeof (char *)))
tutorial(queries)%
```



Check in the file `reasons.h` to the Version Control System:

Figure 4-16 *Checking In a File to the Version Control System*

```
tutorial(queries)% vcs checkin reasons.h
Comment?
> Added another reason why to reasons.h header file
>
<
1.3
1 inserted
0 deleted
11 unchanged
tutorial(queries)%
```

Make sure that the changed component builds:

Figure 4-17 *Building the Changed Component*

```
tutorial(queries)% make
cc -g -o why why.c
tutorial(queries)%
```

**Note:** The `why` program may return a variety of responses.

At this point, run any tests you believe can confirm the results of the build. For example, invoke the program `why` to find out why:

Figure 4-18 *Testing the Changed Component*

```
tutorial(queries)% why
Looks like a feature to me.
tutorial(queries)%
```



Once again, attempt a reconcile of the changed component from the child environment into the parent environment, in this example a reconcile component queries with component example:

Figure 4-19 *Attempt reconcile of Changed Component: Changes in Parent*

```
tutorial(queries)% reconcile :example_comp
Reconciling with example:
Current Comp :example_comp

Determining what objects to reconcile:
Checking: Comp :example_comp
Checking: Comp :example_comp:bin
Checking: Comp :example_comp:bin:why
Checking: Comp :example_comp:scripts
The following objects have changed in example and must be resynced:
File /usr/nse/example/bin/reasons.h
Resyncing:
Comp :example_comp, revision number 3, created 11/05/87 23:11:50 by doe.
Resync conflict: File /usr/nse/example/bin/reasons.h
tutorial(queries)%
```

reconcile automatically invokes resync and indicates that a resolve is necessary. This happens whenever someone makes changes to objects in the component of the parent environment and reconciles their changes to the component with the parent environment before you have reconciled your changes to the component with the parent environment.

When the user runs a shell that has access to a bitmapped screen, the resolve command invokes window- and mouse-based resolve applications. For more information on these applications, see the descriptions of `fileresolve` and `dbresolve` in Section 4.3, *Command Summaries*.

However, this example assumes that the user doesn't have a bitmapped screen. Without a bitmapped screen, resolve performs like this:

Figure 4-20 *Performing a resolve*

```
tutorial(queries)% resolve
The following objects need merging:
File /usr/nse/example/bin/reasons.h
tutorial(queries)%
```



To complete the resolution procedure, you must merge your changes to `reasons.h` with the changes made to `reasons.h` in the parent environment. When you resynced the component with the parent environment, `resync` created `.A` and `.M` files for each file that conflicted with the corresponding files in the parent environment. In the case of this example, list the files to make sure they are there:

Figure 4-21 *Listing Resolution Files*

```
tutorial(queries)% ls reasons.h*
reasons.h          reasons.h.A      reasons.h.M
tutorial(queries)%
```

To make it possible to reconcile the component properly, compare the changed files with their corresponding `.A` and `.M` files. The `.A` files represent the “common ancestor” files, or the original files in the parent environment with which the users in both child environments reconciled their changed files. The `.M` files represent the changed files that the user in the other child environment resolved with the parent environment before you reconciled your changes to the parent environment. In other words, the “common ancestor” `.A` files differ from the `.M` files by the changes that someone else reconciled into the parent environment.

Without the window- and mouse-based `fileresolve` application, the `diff` command is probably the best tool for these file comparisons. Then, using your favorite text editor, edit the changed files so you are sure they have the contents you want to go into the parent environment.

So in the case of this example, use the `diff` command to compare `reasons.h` with `reasons.h.A` and with `reasons.h.M`, edit `reasons.h` so that the file has the contents you want, and mark the file using the merged command:

Figure 4-22 *Marking File as Merged*

```
tutorial(queries)% merged reasons.h
tutorial(queries)%
```

Now, try reconcile again. Congratulations if it succeeds – that means that you resolved properly your changes with the changes other people made in the parent environment. If at first you don’t succeed in reconciling your changes, try, try again until you do.



Someone else may have reconciled some more changes in between the time you first tried to reconcile your changes and the time when you finished your first round of change resolutions. When this happens, cycle through the `reconcile/resync/resolve` process again until your changes merge without conflicts.

For information about how to resolve objects that aren't files, and how to handle resolving objects that the user has renamed, see Chapter 2, *Resolving Objects*.

### 4.3 Command Summaries

**Note:** A standard format for variant names is the output format of the `/bin/arch` command, say `sun3`.

This section contains command summaries for all of the NSE commands accessible through the command-line interface.

When a user types a command name without any arguments or with incorrect options, some of the commands provide a usage summary. When a user types a command name followed by `help`, some of the commands provide help information.

For commands with an environment as an argument, the user may specify the environment name alone, or the environment name with a */variant* suffix. For example, with a command like `activate`, which takes the */variant* suffix to environment arguments, the user can specify environment tools just as `tools`, or the user can specify environment tools with variant `Sun-3` like this:

```
tools/sun3
```

When a user specifies the */variant* suffix to an environment argument of a command that doesn't permit the */variant* suffix specification, the command provides an error.

Some commands take optional username and machine specifications for environments as well. These specifications may be necessary to clear up any ambiguity between environments with the same base name. For example, to disambiguate environment `env7` owned by user `doe` from environment `env7` owned by user `ray`, a user could specify environment `env7`, owned by user `doe`, and registered on machine `tutorial`, like this:

```
env7@doe@tutorial
```

In the case of public environments, the owner's username is `Nse`, the NSE administration username, and the commands ignore any machine specification provided by the user. So, a user could specify public environment `pubenv3`, owned by the NSE administration username `Nse`, like this:

```
pubenv3@Nse
```

Some commands require that only the owner of an environment can invoke that command on the environment. However, the administration username `Nse` can invoke these commands on environments even when `Nse` doesn't own the environment. In addition, the administration username `Nse` is the only username that can invoke certain other administration commands.

The rest of the chapter contains the command summaries.



acquire

```
acquire [-c childenv] [-m] [-n] [-q] [-r rootdir]
        [-v variant] ... [-x childenv] [-C] parentenv
        component ...
```

or

```
acquire -R [-q] [-x childenv]
```

`acquire` obtains the latest instance of an NSE component or components from a parent environment for use within a child environment. From within an activated child environment, the user invokes `acquire` without the `-c` or `-x` option to acquire a component or components to that child environment. When the child environment already exists, but the user isn't executing the command within an activated instance of the child environment, the user can acquire with the `-x` option to "extend" that child environment with the specified component or components. To create a child environment and acquire a component or components to that newly created child environment, use `acquire` with the `-c` option. When creating a child environment with the `-c` option, the user must specify any desired variants with the `-v` option, or `acquire -c` creates the environment with the default variant associated with the current machine architecture.

An `acquire` command can result in conflicts on objects just as a `resync` command can. Since users typically invoke `acquire` to acquire new components, the likelihood of producing conflicts is less than the likelihood of producing conflicts with `resync`, which the user invokes to update old components. See the description of the `resync` command for more information on conflicts.

The syntax for component names consists of a root component, specified by a colon (`:`), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

**Note:** When specifying an open bracket on the command line, remember to preface it with a backslash character (\), so that the shell doesn't misinterpret it.

`acquire` obtains the latest revision of a component or components, unless the user specifies a revision name in square brackets appended to a component name. For example, to acquire the revision named `rev1` of the component named `:src:bin:cat`, the user specifies the following *component* to the `acquire` command:

```
:src:bin:cat [rev1]
```

When the user appends a revision name to a component name, `acquire` obtains a "consistent" instance, in other words that named revision of the component.

**Note:** When acquiring the latest consistent revision of a component using the empty square bracket syntax, the user need not know the name of the revision to acquire it.

To obtain the latest "consistent" instance, in other words the latest revision, of a component, the user may specify the component name, appending empty square brackets after it. For example, to obtain the latest revision, say `rev5`, of the component named `:src:bin:cat`, the user could specify:

```
:src:bin:cat [ ]
```

When the user specifies the abbreviation colon (:) as the *component* argument to `acquire`, the command acquires all of the components from the specified environment.

When the user specifies an environment argument to `acquire`, `acquire` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

#### Options:

`-c childenv`

Create new child environment, which runs unactivated. From within an activated child environment, the user invokes `acquire` without the `-c` or `-x` option to acquire a component to that child environment.

When invoked by the administrative username `Nse`, the `acquire -c` command creates a public environment. Public environments are not accessible to users other than `Nse` until the yellow pages propagates news of their existence; usually, this takes about an hour after creation of the public environment.



- `-m` Don't automatically update objects that have changed in the component or components located in the parent environment, but haven't changed in the component or components located in the child environment. Instead, **make** a conflict on such objects. The default is to update automatically these objects.
- `-n` Do **nothing**, just indicate what would be done.
- `-q` Quiet execution prevents command from printing out any messages, except error messages.
- `-r rootdir` Create the environment using disk space in a directory under the root directory named *rootdir*. Because the NSE stores the environment in a directory *under* the root directory, the user can reuse the same root directory name for more than one environment's root directory. By default, the NSE stores environments in the directory `~/.nse_data`.
- When the user wants users on other machines to be able to activate the environment, the user must locate the environment on an exported file system directory, particularly when the root directory is on the local disk of a user's machine.
- The `-r` option is valid only when used along with the `-c` option.
- `-v variant` Acquire variant *variant*. Use the `-v` option only when creating a child environment with the `-c` option. When you don't use the `-v` option along with the `-c` option, `acquire -c` creates an environment

`-x childenv`

with the default variant associated with the current machine architecture.

Extend the existing child environment called *childenv*, which is not currently activated. From within an activated child environment, the user invokes `acquire` without the `-c` or `-x` option to acquire a component to that child environment.

`-C`

Copy files to the current environment rather than trying to share them with *parentenv*.

`-R`

Recover, in other words, cancel an interrupted `acquire`. Doesn't work with any other options except the `-q` and `-x` options. This option may actually acquire other objects to bring the environment to a consistent state.



activate

activate [-r] env[/variant] [command arg ...]

activate makes an NSE environment accessible to the user. When the user specifies the *command* argument, activate runs the command activated in the specified environment. Otherwise, activate starts up an activated shell for the specified environment.

In both cases, activate assigns the shell environment variables with the values from the variant specification for this activated environment, set by the `nsevar assign` command.

To activate an environment that has no children so that the user can read from and write to the environment, the user simply activates that environment. When the user activates an environment so that user can read from and write to the environment, no user can reconcile to that environment until the user who activated the environment unactivates it.

To activate an environment that has children so that the user can read from and write to the environment, the user must place a "lock" on the environment before activating it, so that other users can't make changes to the environment during the activation. The `nseenv lock` command locks environments; `nseenv unlock` unlocks an environment so other users can make changes to that environment once again.

When the user makes changes to a locked environment that has children, the user should preserve the changed components before unlocking that environment; otherwise, the user may lose some of the changes.

When the user activates an environment that has children without first locking that environment, the environment activates read-only, just as when the user specifies the `-r` option.

To deactivate an environment, type CTRL-D to the activated environment command prompt.

The user can't delete an environment until after deactivating the environment.

When an environment is an argument to activate, the environment can appear either as an environment name alone, or with a /*variant* suffix. For example, the user can specify environment tools just as:

tools

**Note:** Although other users can't make changes to an environment that a user has locked, they can still browse the locked environment.

or the user can specify environment tools with the Sun-3 variant sun3 like this:

```
tools/sun3
```

A standard format for variant names is the output format of the arch command, for example, sun3. For more information about the arch(1) command, see the Man Page online or in the *Commands Reference Manual*.

|         |    |                                                                                                                                                                                                                                                               |
|---------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option: | -r | <p>Specify read-only activation; in other words, users in read-only activated environment cannot make changes to development objects within that environment.</p> <p>The default activation of environments that have children is a read-only activation.</p> |
|---------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



`append`

```
append [-a component] [-m] [-n] [-q] [-t type]  
[-x childenv] [-C] parentenv object...
```

or

```
append -R [-q] [-x childenv]
```

By default, `append` acquires the latest instance of an object or objects of type `File` from *parentenv* to the current environment. When specifying a file to append, the user must provide the fully qualified name, in other words, the absolute pathname of the file.

To append objects of type other than `File`, the user must specify the object type with the `-t` option.

When specifying a link database object to append, the user must provide the fully qualified name, in other words, the absolute pathname of the link database.

When specifying a target object to append, the user must provide a target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname for the Makefile associated with the target.

An `append` command can result in conflicts on objects just as a `resync` and `acquire` command can. Since users typically invoke `append` to append new objects, the likelihood of producing conflicts is less than the likelihood of producing conflicts with `resync`, which the user invokes to update old components. See the description of the `resync` command for more information on conflicts.

To append “consistently” the revisions of objects, the user must specify the `-a` option which appends instances of objects current as of the revision time of the latest revision of the *component* argument to the `-a` option. The user may specify earlier revisions in the *component* argument using the double square bracket syntax described below.

The syntax for component names consists of a root component, specified by a colon (:), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

**Note:** When specifying an open square bracket on the command line, remember to preface it with a backslash character (`\`), so that the shell doesn't misinterpret it.

The user may also add a revision name suffix, surrounded by an open square bracket (`[`) and a close square bracket (`]`), to the name of the component. For example, the user can specify the `rev7` revision of the example component above by typing:

```
:src:bin:cat [rev7]
```

When the user specifies an environment argument to `append`, `append` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

|          |                           |                                                                                                                                                                                                                                                                                                                    |
|----------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Options: | <code>-a component</code> | Append “consistently” instances of the specified object or objects current as of the revision time of the latest, or of the specified revision of <i>component</i> .                                                                                                                                               |
|          | <code>-m</code>           | Don't automatically update objects that have changed in the component or components located in the parent environment, but haven't changed in the component or components located in the child environment. Instead, <b>make</b> a conflict on such objects. The default is to update automatically these objects. |
|          | <code>-n</code>           | Do <b>nothing</b> , just indicate what would be done.                                                                                                                                                                                                                                                              |
|          | <code>-q</code>           | Quiet execution prevents command from printing out any messages, except error messages.                                                                                                                                                                                                                            |
|          | <code>-t type</code>      | When appending an object or objects of <i>type</i> other than <code>File</code> , the user must specify the object <i>type</i> .                                                                                                                                                                                   |
|          | <code>-x childenv</code>  | Extend existing child environment called <i>childenv</i> , which is not currently activated. From within an activated child environment, the user invokes <code>append</code> without the <code>-x</code> option to append a component to that child environment.                                                  |



- C Copy files to the current environment rather than trying to share them with *parentenv*.
- R Recover, in other words, cancel an interrupted append. Doesn't work with any other options except the -q and -x options.

## bootstrap

## bootstrap

`bootstrap` either creates an initial, or extends an existing NSE development object hierarchy. The command prompts the user with a series of questions.

When the user creates a new object hierarchy, `bootstrap` prompts the user for the environment name, the directory within which the environment should reside, additional variant names for the environment, control points for the environment, covered directories within the environment, and the full pathname of the component description file, if it already exists, or the full pathname of the top-level Makefile, and the name of the top-level target, if the component description file doesn't exist. In addition, `bootstrap` asks the user to specify whether or not to copy the source control history of the environment contents.

Newly created environments are not accessible to users other than the owner of the environment until the Yellow Pages propagates news of their existence; usually, this takes about an hour after creation of the environment.

The administrative user `Nse` cannot invoke the `bootstrap` command. When a user wants to create a public environment, the user must complete the bootstrap process; then the administrative user `Nse` can invoke the `nseenv public` command to transform the private environment created by `bootstrap` into a public environment.

When the user extends an existing object hierarchy, `bootstrap` prompts the user for the environment name, the name of the component to extend, and the full pathname of the component description file, if it already exists, or the full pathname of the top-level Makefile, and the name of the top-level target, if the component description file doesn't exist.

`bootstrap` performs several preserve and acquire operations. When the user doesn't specify a component description file, `bootstrap` invokes the `describe_comps` command to obtain one. The name of the default component description file produced by `describe_comps` is `NSE.comps`.

Should a `bootstrap` command fail, the user can attempt a recovery of the `bootstrap` command by invoking the command once again.

Because an environment can have many variants, the `bootstrap` command doesn't bootstrap derived files. To



obtain the derived files after bootstrap, execute a build of the components using `make`. For more information about `make`, see the SunPro *make User's Guide*, or the *Programming on the Sun Workstation* manual.

The syntax for component names consists of a root component, specified by a colon ( : ), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

The user may not specify a revision name for the component the user specifies when prompted by `bootstrap`.

When the user specifies an environment to a `bootstrap` prompt, `bootstrap` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

For more information about the bootstrap process, see the *Network Software Environment: Installation and Administration* manual.

other words, if Auto Advance is set, do not advance; when Auto Advance is not set, advance.

CTRL-Left           Accept the left-hand side for the current difference, and for all subsequent differences up to the end of the object.

Right                Accept the right-hand side of the current difference. When Auto Advance is in effect, advance to the next difference.

SHIFT-Right        Accept the right-hand side and then do the opposite of the current Auto Advance setting.

CTRL-Right         Accept the right-hand side for the current difference, and for all subsequent differences up to the end of the object.

Reject               Reject the changes for the current difference.

SHIFT-Reject        Reject the changes for the current difference and then do the opposite of the current Auto Advance setting.

CTRL-Reject         Reject the changes for the current difference, and for all subsequent differences up to the end of the object.

Undo                Undo the last Left, Right, or Reject operation. You can't undo an undo.

SHIFT-Undo         Undo all the operations on this object.

Scroll-Lock         When in effect, the three text-subwindows scroll in unison. Otherwise each subwindow scrolls independently.

Unresolved Diffs:

Shows the number of outstanding unresolved differences. As the user resolves differences by applying the Left , Right, or Reject buttons, this number decreases. As the user undoes resolutions by using the Undo button, this number increases.



## Property Sheet

You can customize dbresolve using the property sheet to set or alter various display and control options. To bring up the property sheet, press the Props function key, typically L3, while the mouse is over any part of dbresolve. The property panel items have the following meanings:

Auto Advance      Advance to the next difference after each Left, Right, or Reject operation. Default is on.

Auto Merge      dbresolve will automatically resolve any differences that do not conflict with differences from the other object. This in effect takes the union of the two sets of changes and left unresolved are only those changes which must be resolved by hand. Default is on. Switching the default to Off and then pressing Apply will cause the current object to be re-read without automerging.

Top Lines      Number of lines in the top two subwindows. Default is 20 lines.

Bottom Lines      Number of lines in the bottom subwindow. Default is 20 lines.

Columns      Number of columns in the left, and in the right subwindow. Default is 70 columns.

Apply      Any changes you have made to the property sheet will now take effect.

Reset      Reset the property sheet to the state it had at the time of the last Apply.

Done      Close the property sheet.

File      ~/.dbresolverc      File storing property sheet information.

See Also      fileresolve, resolve, and textedit

Bugs      Using the Find function key causes the subwindows to get out of sync for scrolling. To resync them, turn Scroll-Lock first off, and then on.

`describe_comps``describe_comps [-o outputfile] target%makefile`

As part of the bootstrap procedure, `describe_comps` generates an NSE component hierarchy description from a Makefile target, tracing down the hierarchy of objects described in that Makefile. The name of the Makefile target is in the form:

*target%makefile*

where *target* is a target in *makefile* that depends on all the files the user wants to bootstrap. Make sure to list all dependencies for this target in *makefile*. The user must specify *makefile* using the fully qualified name, in other words the absolute pathname for the Makefile.

Within the component description, the name of the “root” component in the component hierarchy is the simple name of the Makefile directory name. For example, executing:

```
describe_comps tools%/usr/src/Makefile
```

produces a component hierarchy with a root component called `src`.

For `describe_comps` to translate Makefiles correctly into a component description, the user must make sure that neither source nor header files appear as targets within Makefiles. Only names of derived files, or other names that aren’t source or header filenames, should appear as Makefile targets. The NSE won’t trace source, header, and derived files appropriately unless the user converts the Makefiles properly when using `describe_comps` as part of the bootstrap procedure. `describe_comps` lists warnings into a file called `NSE.comps.warnings` when it detects what may be a source or header file listed as a Makefile target.

As a default, `describe_comps` produces a *componentfile* called `NSE.comps` in the directory from which the user invokes the command.

The syntax for component names consists of a root component, specified by a colon ( : ), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

`:src:bin:cat`



Option:            `-o outputfile`        Output to *outputfile*. Default is  
NSE.comps.

## fileresolve

fileresolve *inputfile*

*fileresolve* is a window-based program for sequentially comparing and merging text files. It displays two files for side-by-side comparison, each in a read-only text-subwindow.

*fileresolve* shows the differences between a file in a child environment (left) and a file in a parent environment (right) with respect to a common (unseen) ancestor file. Beneath them, users can edit in another subwindow to construct a *merged* version – one which contains selected lines from either or both input files, along with any additional edits the user makes.

Only the *resolve* command invokes *fileresolve*; the user should never invoke *fileresolve* directly.

## Glyphs

The text in the editing subwindow, or *outfile*, initially contains the lines that are in common between *leftfile* and *rightfile* and blank lines as place-holders for those lines that differ.

Each of the three text-subwindows contains a region where *fileresolve* places glyphs to describe the differences.

In the two read-only text-subwindows, containing *leftfile* and *rightfile*, *fileresolve* uses three glyphs – a plus sign ( + ) for added lines, a minus sign ( - ) for deleted lines, and a vertical bar ( | ) for changed lines. In the editing subwindow containing *outfile*, *fileresolve* indicates place-holder lines using a small dot. After the user resolves a difference, the glyphs become hollow.

## Resolution of Differences

At any given time, one of the displayed “differences” is *current*. *fileresolve* indicate the current difference by displaying its glyphs in reverse video. When constructing a merged version, *fileresolve* keeps track of whether or not a difference has been “resolved.” *fileresolve* considers a difference resolved when the user either applies the left or right side, or applies the By Hand button to the difference. The user can resolve most differences with the Left and Right buttons; however, the resolution of some differences requires hand-editing. The user can edit the text of *outfile* directly with the *textedit* commands. In these cases, the user can inform *fileresolve* that the differences are resolved by clicking on the By Hand button



## Command Buttons

Many of the command buttons have menu items associated with them. The user can access the menu items by depressing the mouse menu button after moving the mouse over the command button, or by applying a modifier, such as depressing the **SHIFT** and/or **CTRL** key, while clicking the command button. The descriptions below show the modifiers necessary to access the functions.

**Commit**                      Equivalent of **Save**, **Checkin**, **Next** – save changes, check file in to Version Control System, and proceed to next file. When a user has already checked out the file, **fileresolve** omits the **Checkin** step.

**SHIFT-Commit**            Equivalent of **Save**, **Checkin** – do not go to next file (see **Skip**). When a user has already checked out the file, **fileresolve** omits the **Checkin** step.

**CTRL-Commit**            Equivalent of **Save** – only save the changes.

**SHIFT-CTRL-Commit**

Equivalent of **Accept Parent** – accept the parent environment's version of the file.

**Skip**                        Do not commit any changes, but proceed to the next file requiring merging. A menu behind this button allows the user to skip to any file in a list of files that need merging.

**Quit**                      Exit the tool. The user must explicitly save the merged *outfile* with the **Save** button. When the user has unsaved changes, **fileresolve** prompts the user for confirmation.

**Next**                      Make the next difference current. The subwindow scrolls, if necessary, to display it.

**CTRL-Next**                Make the next unresolved difference current.

**SHIFT-CTRL-Next**

Make the next resolved difference current.

**META-Next**                Make the next difference on the left current.

**SHIFT-META-Next**

Make the next difference on the right current.

## dbresolve

dbresolve *action object diff\_prog apply\_prog inputfile*

dbresolve is a window-based program for sequentially comparing and merging NSE components and other objects. Like *fileresolve*, dbresolve displays the object in the child environment on the left side, and the object in the parent environment on the right side. Unlike *fileresolve*, which shows each file in its entirety, dbresolve displays only the changes made to the object. Beneath these two windows, a third text window displays the changes the user has accepted. When the user commits the changes in the third text window, dbresolve applies the changes to the object that results from the merge. Unlike *fileresolve*, the user can't edit the contents of the third text window.

Only the *resolve* command invokes dbresolve; the user should never invoke dbresolve directly.

### Glyphs

Each of the three text-subwindows contains a region where dbresolve places glyphs to describe the differences. In the two side-by-side text-subwindows three glyphs are used – a plus sign ( + ) for added records, a minus sign ( - ) for deleted records, and a vertical bar ( | ) for changed records. In the bottom subwindow, a small dot indicates place-holder records. After the user resolves a difference, the glyphs become hollow.

### Resolution of Differences

At any given time, one of the displayed “differences” is *current*. The current difference is indicated by displaying its glyphs in reverse video. When constructing the merged version, dbresolve keeps track of whether or not a difference has been “resolved.” dbresolve considers a difference resolved when the user accepts either the *Left* or the *Right* side, or when the user rejects one or both changes with *Reject*. Unlike *fileresolve*, dbresolve only accepts or rejects changes; the user can't edit the objects by hand.

### Command Buttons

Many of the command buttons have menu items associated with them. The user can access the menu items by depressing the mouse menu button after moving the mouse over the command button, or by applying a modifier, such as depressing the *SHIFT* or *CTRL* key, while clicking the command button.



The descriptions below show the modifiers necessary to access the functions.

**Commit**                      Commit the series of accept and reject actions for this merge and move on to the next object. If there are unresolved differences, dbresolve prompts the user for confirmation.

**SHIFT-Commit**              Equivalent of **Accept Parent** – accept the parent environment's version of the object, in other words, the changes on the right.

**Skip**                          Do not commit any changes, but proceed to the next object requiring merging. When there are unresolved differences, the user must confirm the operation.

**Quit**                         Exit the tool. When the user has uncommitted changes, the user must confirm the operation.

**Next**                         Make the next difference current. The subwindow scrolls, if necessary, to display it.

**SHIFT-Next**                 Make the next unresolved difference current.

**CTRL-Next**                 Make the next resolved difference current.

**Prev**                         Make the previous difference current. The subwindow scrolls, if necessary, to display it.

**SHIFT-Prev**                 Make the previous unresolved difference current.

**CTRL-Prev**                 Make the previous resolved difference current.

**Goto**                         Make the difference containing the selection current. When the selection is not on a line of a difference, dbresolve chooses the first difference following the selection.

**Left**                         Accept the changes on the left-hand side of the current difference. When **Auto Advance** is in effect, advance to the next difference.

**SHIFT-Left**                 Accept the left-hand side and then do the opposite of the current **Auto Advance** setting. In

- Prev** This button and all of its keyboard variations are as with **Next** above, except that in each case **fileresolve** makes the previous item current.
- Goto** Make the difference containing the current selection current. When the selection is not on a line of a difference, **fileresolve** chooses the first difference following the selection.
- Left** Apply the left-hand version of the current difference to *outfile*. When **Auto Advance** is in effect, advance to the next difference.
- SHIFT-Left** Apply the left-hand version and then do the opposite of the current **Auto Advance** setting. In other words, when **Auto Advance** is set, do not advance; when **Auto Advance** is not set, advance.
- CTRL-Left** Apply the left-hand version for the current difference, and for all subsequent differences up to the end of the file.
- Right** Apply the right-hand version of the current difference to *outfile*. When **Auto Advance** is in effect, advance to the next difference.
- SHIFT-Right** Apply the right-hand version and then do the opposite of the current **Auto Advance** setting.
- CTRL-Right** Apply the right-hand version for the current difference, and for all subsequent differences up to the end of the file.
- By Hand** Inform **fileresolve** that the user has resolved the current difference by hand.
- SHIFT-By Hand** Inform **fileresolve** that the current difference has been resolved by hand and then do the opposite of the current **Auto Advance** setting.
- CTRL-By Hand** Inform **fileresolve** that the current difference and all subsequent differences up to the end of the file have been resolved by hand.
- Undo** Undo the last **Right** or **Left** operation. You can't undo an undo.
- SHIFT-Undo** Undo the last operation applied to the current difference.



CTRL-Undo      Undo all the operations since the last Load.

Scroll-Lock      When in effect, the three text-subwindows scroll in unison. Otherwise each subwindow scrolls independently.

Unresolved Diffs:

Shows the number of outstanding unresolved differences. As the user resolves the differences by applying the Left, Right, or By Hand buttons, this number decreases. As the user undoes resolutions using the Undo button, this number increases.

### Property Sheet

You can customize `fileresolve` using the property sheet to set or alter various display and control options. To bring up the property sheet, press the Props function key, typically L3, while the mouse is over any part of `fileresolve`. The property panel items have the following meanings:

Auto Advance      Advance to the next difference after each Left, Right, or Reject operation. Default is on.

Auto Merge      `fileresolve` automatically resolves any differences in *leftfile* or *rightfile* that do not conflict with differences from the other file. Default is on. Switching the default to Off and then pressing Apply will cause the current object to be re-read without automerging.

Top Lines      Number of lines in the top two subwindows. Default is 20 lines.

Bottom Lines      Number of lines in the bottom subwindow. Default is 20 lines.

Columns      Number of columns in the left, and in the right subwindow. Default is 70 columns.

Apply      Any changes you have made to the property sheet will now take effect.

Reset      Reset the property sheet to the state it had at the time of the last Apply.

Done      Close the property sheet.

File      `~/ .filemergerc`      File storing property sheet information.

- See Also**      `resolve`, `filemerge(1)`, `diff(1)`, `sdiff(1)`, and `textedit(1)`.
- Bugs**          Using the `Find` function key causes the subwindows to get out of sync for scrolling. To resync them, turn `Scroll-Lock` first off, and then on.
- `fileresolve` doesn't permit use of the `textedit` function key called `Undo`.



`linksvc``linksvc [database]`

The Link Service is a facility for linking together entities within various applications. For more information, see Chapter 3, *Links and the Link Service*.

lvs

lvs *command arg ...*

*lvs* is the NSE version control system for link databases.  
The user must check out link databases before modifying them.

Commands: *lvs* requires the user to specify one of these commands:

*checkin* [-c *comment*] [-q] *linkdb ...*

Check in a version of the link database named *linkdb* to save all the changes necessary for the next version.  
Preserves the source link database from which the version is made.

Options:

- c *comment* Specify comment associated with version user checks in. Enclose *comment* in quotes when there are space characters in it.
- q Quiet execution prevents command from printing out any messages, except error messages.

*checkout* [-q] *linkdb ...*

Check out the latest version of *linkdb* for use in development, for example, editing.

Option:

- q Quiet execution prevents command from printing out any messages, except error messages.

*ci* [-c *comment*] [-q] *linkdb ...*

Same as *checkin*.

*co* [-q] *linkdb ...* Same as *checkout*.

*create* [-c *comment*] [-q] *linkdb ...*

Place *linkdb* under VCS control.  
Preserves the source link database from which the version is made.



## Options:

- c *comment* Specify comment associated with version user creates. Enclose *comment* in quotes when there are space characters in it.
- q Quiet execution prevents command from printing out any messages, except error messages.

hi [-v *version*] *linkdb*

Same as *history*.

history [-v *version*] *linkdb*

Print VCS history associated with *linkdb*.

## Option:

- v *version* Print history of version numbered *version*. Example version number is:  
1.3

rename [-q] *linkdb1 linkdb2*

Change the name of *linkdb1* to *linkdb2*.

## Option:

- q Quiet execution prevents command from printing out any messages, except error messages

uncheckout [-q] *linkdb*

Undo a previous *lvcs* checkout, discarding any changes made since that checkout.

## Option:

- q Quiet execution prevents command from printing out any messages, except error messages.

merged

merged [-a] [-f] [-t *type*] *object* ...

or

merged -s [-t *type*] *object object.M*

or

merged -p

The merged command marks as “merged” any objects resync, acquire, or append reports conflicts on.

The merged command is primarily for users who don’t have access to bitmapped screens, and therefore can’t use the Browser or the resolve command, both of which require a bitmapped screen.

As part of resolving the differences between the contents of a child environment and the contents of its parent environment, the user must merge, and mark as “merged,” any objects that resync, acquire, or append report conflicts on. resync, acquire, or append report conflicts on these objects because users have changed the objects in both the parent and child environments. The user must run the merged command to mark objects “merged” that the user wants the NSE to treat as merged for the purpose of the reconcile command.

Marking the appropriately merged objects as “merged” ensures that the NSE contains the appropriate versions of the development objects in the parent and child environments. In most cases, the user reconciles again the component in the child environment with the component in the parent environment after completing the merging and marking procedure.

The default object type to mark “merged” is File, but the user may specify that merged mark other types of objects, including components, using the -t option. It is necessary to specify the object’s type because objects of different types may have the same name.

When specifying a file or link database, the user must provide, the fully qualified, in other words, absolute pathname for that file or link database.

When specifying a target, the user must provide the target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname of the Makefile associated with the target.



The syntax for component names consists of a root component, specified by a colon ( : ), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. An example component name is:

```
:src:bin:cat
```

The user may not specify a revision name of any *component* arguments of the merged command.

merged won't mark unstable objects as "merged" unless the user specifies the `-f` option to force marking as "merged." An example of an unstable object is a file that a user has checked out of, but not checked back in to the version control system.

When the user wants to accept the version of the object or objects from the parent environment, the user must copy the parent environment's version of the object or objects into the corresponding object or objects in the child environment and mark them using the `-a` option of the merged command.

The reason for the `-a` option to the merged command is that the subsequent reconcile occurs more efficiently because the user has marked the contents of the parent as the result of the merge.

In the case of files, `resync` has already copied the parent environment's version of the files into the child environment, tagging the parent environment's files with `.M` suffixes. The user must replace the child environment files with the corresponding `.M` files that `resync` acquired into the child environment. In other words, for a file called `file1` in the child environment, `resync` acquires a file named `file1` in the parent environment to a file named `file1.M` in the child environment, and the user must copy `file1.M` over `file1` in the child environment before marking `file1` as merged with the `-a` option.

Once the user marks an object as merged with the `-a` option, the user reconciles once again more efficiently the component in the child environment with the component in the parent environment. The reconcile following the marking of the merged object or objects is more efficient because `reconcile` need not copy the object or objects from the child environment back into

the parent environment once the user has marked the object or objects using the `-a` option of the `merged` command.

Similar to the `-n` option of the `resolve` command, the `-p` option of `merged` prints out the conflicts that the user still needs to resolve between the contents of the component in the child environment and the contents of the component in the parent environment.

|          |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Options: | <code>-a</code>                           | Accept the version of the object or objects in the parent environment.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|          | <code>-f</code>                           | Force marking objects, including unstable objects as "merged." An example of an unstable object is a file that a user has checked out of the version control system.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|          | <code>-p</code>                           | Print out (list) the conflicts that the user still needs to resolve between the contents of the component in the child environment and the contents of the component in the parent environment.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|          | <code>-s [-t type] object object.M</code> | Separate the two versions of the same object resulting from an acquire or resync id conflict into two distinct objects, substituting <code>object.M</code> , the version of the object with a <code>.M</code> suffix, for the current version of <code>object</code> in the child environment.<br><br>When acquire or resync detects an id conflict on <code>object</code> , the command copies <code>object</code> from the parent environment into <code>object.M</code> in the child environment.<br><br>Acquire and resync warn about id conflicts with a message that lists the name of the object. These warning messages indicate that the object in the parent environment and the object in the child environment are registered as two separate objects with the same name. This can happen when the user bootstraps the same object twice, or |



when the user copies the object into the child environment using the `cp` command, instead of acquiring the object from the parent environment. Usually, the user resolves the differences between these objects manually or by using the `resolve` command.

However, in rare cases, the user may want to keep the object in the child environment as an object distinct from the object in the parent environment. This can happen when the user creates a new object with the object name of an object that already exists in the parent environment. In this case, the user probably doesn't want to merge the two objects, so the user can specify `merged -s` to "separate" the two versions of the object into two distinct objects.

`merged -s` registers the `object.M` object as the object corresponding to the object `object` in the parent environment.

At this point, the user must decide whether to rename `object`, `object.M`, or both in the child environment using the `rename` command for the object's type (`vcs rename` for `File` type). In the example described so far, the user would probably rename `object` to some other object name, and rename `object.M` to `object`. The user should ensure that all references to `object` in Makefiles and source files reflect the new name for `object`.

`-t type`

Type of object to mark merged.  
Necessary to specify `type` of object except when object is the default `File` type.

## nsebrowse



**Note:** In the case of the administrative username `Nse`, the list of environments owned by the user and the list of public environments are identical.

## nsebrowse

The `nsebrowse` command starts up the NSE Browser, a window- and mouse-based interface to the NSE. The Browser provides a variety of operations for the examination and manipulation of the software development object hierarchy. The Browser starts by displaying the list of environments owned by the user. When the user doesn't own any environments, the Browser the list of public environments. With its menus and display, the Browser facilitates user searches through environments and the component hierarchies associated with them. The Browser saves a list of the queries the user makes, so that the user can repeat them as desired. The Browser supports nearly all of the functionality of the command-line interface.

## Browser Subwindows:

The NSE Browser window is divided into three subwindows:

- The Legend subwindow
- The Display subwindow
- The Log subwindow

The Legend subwindow shows the current location within the environment and component hierarchies. The Display subwindow shows the results of Browser queries, such as the component hierarchy for the current environment. The Log subwindow shows a log of commands executed as a result of Browser operations.

## Browser Symbols:

These are the symbols used in the Browser:

- Environment
- Component
- make Target
- File

The environment symbol always appears in the Browser Legend to the left of the word `Environment`. The Browser environment symbol may appear in the Display as well.

The component symbol appears in the Legend to the left of the word `Component`, when a user is listing the components of an environment. In most cases, the name of the environment containing the listed components appears in the box to the right of the word `Environment`. The Browser component symbol may appear in the Display as well to show the component hierarchy.



### Browser Menus and Operations:

The Browser provides a variety of operations accessible through menus, which are similar to many of the commands in command-line interface. For more information, see Chapter 1, *The Browser*.

nsecomp

nsecomp [-e *env*] [-i] *command arg ...*

The *nsecomp* command manipulates NSE components.

When the user invokes *nsecomp* within an activated environment, *nsecomp* assumes that environment is the focus of the *nsecomp* command, unless the user changes the environment focus using the *-e* option of the *nsecomp* command, or using the *nsecomp set* command in interactive sessions or script files (described below).

When the user invokes *nsecomp* outside of an activated environment, the user must specify the environment focus with the *-e* option or an error results.

The syntax for component names consists of a root component, specified by a colon (:), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

When the user specifies an environment argument to *nsecomp*, *nsecomp* returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

Options:            *-e env*            Set environment in which to execute commands to *env*.

*-i*            Interactive session.

Commands:        Users can specify these commands in one of these ways:

- 1) On the *nsecomp* command line.
- 2) In a file of commands, in other words, a script file, provided as standard input to *nsecomp*. For example, the user can specify a script file called *scriptfile* to the *nseenv* command by typing:

```
nsecomp < scriptfile
```

- 3) In an interactive session. The *-i* option to *nsecomp* starts an interactive session.

add *comp* [*type object ...*]

Add an object to the specified component in the current environment. In most cases, the user specifies the



object named *object* of type *type* to add to the component named *component*. For example, to add the target named `all%/src/bin/cat/Makefile` to the component named `:src:bin:cat`, use the command:

```
nsecomp add :src:bin:cat Targ all%/src/bin/cat/Makefile
```

The user may specify more than one object of type *type* to add to the component.

When the user adds an object to the component, the user doesn't create the object. In fact, the user can add to a component an object that doesn't exist.

When the user specifies a target as object, the user should specify the target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname of the Makefile associated with the target.

When the user specifies a file or link database as *object*, the user should specify the fully qualified, in other words, absolute pathname of the file or link database. An object of type `File` or type `Linkdb` must be under a control point of the current environment.

In the case of addition of a subcomponent to the component, the user may specify the fully qualified name of the subcomponent named *component* as the only argument. For example to add the subcomponent `dog` to the component named `:src:bin:cat`, use the command:

```
nsecomp add :src:bin:cat:dog
```

help

List help information for nsecomp.

history [-l] [*component* ...]

List the revision history information for the root component, or the optionally specified *component* or components in the current environment.

Option:

-l

Long listing, including the comments, of the revision history information for the component called *component* in the current environment. Same output format as the `nseenv history -l` command.

list [-l] [-r] [*component* ...]

List the contents of the root component, or the optionally specified *component* or components in the current environment.

Options:

-l

Long listing which gives the revision names.

-r

Recursive listing which lists what the components contain, similar to the `-r` option of the SunOS `ls` command.

quit

Exit interactive session.

remove *component* [*type object* ...]

Remove an object from the specified *component* in the current environment. In most cases, the user specifies the object named *object* of type *type* to remove from the component named *component*. For example, to remove the target named `all%/src/bin/cat/Makefile` from the component named `:src:bin:cat`, use the command:

```
nsecomp remove :src:bin:cat Targ all%/src/bin/cat/Makefile
```

The user may specify more than one object of type *type* to remove from the component.



With the exception of removal of a component, when the user removes an object from a component, the user doesn't delete the object. In fact, any existing object except for components continues to exist after removal with the `nsecomp remove` command.

When the user specifies a target as object, the user should specify the target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname of the Makefile associated with the target.

When the user specifies a file or link database as *object*, the user should specify the fully qualified, in other words, absolute pathname of the file or link database.

In the case of removal of a subcomponent from a component, the user may specify the fully qualified name of the subcomponent named *component* as the only argument. For example to remove the subcomponent `dog` from the component named `:src:bin:cat`, use the command:

```
nsecomp remove :src:bin:cat:dog
```

```
rename fromcomp tocomp
```

Rename component *fromcomp* as component *tocomp*.

```
set env
```

Use environment *env* as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default environment using the `nsecomp set` command.

nsecontrolpt

nsecontrolpt [-e *env*] [-i] *command arg ...*

The `nsecontrolpt` command manipulates the control points of NSE environments. A control point is a directory that represents a portion of the SunOS file system hierarchy that the user makes accessible within an environment. Most environments have only a few control points.

When the user creates new control points for an environment, the new control points only exist in environments that the user activates after creating the new control points. The user should not make changes to control points associated with an environment while activated in that environment.

Only the owner of the environment and the administrative user `Nse` can change the set of control points associated with an environment.

When the user specifies an environment argument to `nsecontrolpt`, `nsecontrolpt` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes `nsecontrolpt` within an activated environment, `nsecontrolpt` assumes that environment is the focus of the `nsecontrolpt` command, unless the user changes the environment focus using the `-e` option of the `nsecontrolpt` command, or using the `nsecontrolpt set` command in interactive sessions or script files (described below).

When the user invokes `nsecontrolpt` outside of an activated environment, the user must specify the environment focus with the `-e` option or an error may result.

The NSE doesn't support a way to delete control points.

For more information on control points, see Section 3.2, *Control Points*, of the *Network Software Environment: Installation and Administration* manual.

|          |                     |                                                              |
|----------|---------------------|--------------------------------------------------------------|
| Options: | <code>-e env</code> | Set environment in which to execute commands to <i>env</i> . |
|          | <code>-i</code>     | Interactive session.                                         |

|           |                                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------------|
| Commands: | Users can specify these commands in one of these ways:                                                                 |
|           | 1) On the <code>nsecontrolpt</code> command line.                                                                      |
|           | 2) In a file of commands, in other words, a script file, provided as standard input to <code>nsecontrolpt</code> . For |



example, the user can specify a script file called *scriptfile* to the *nsecontrolpt* command by typing:

```
nsecontrolpt < scriptfile
```

- 3) In an interactive session. The *-i* option to *nsecontrolpt* starts an interactive session.

|                         |                                                                                                                                                                                                                          |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>add controlpoint</i> | Make the part of the SunOS file system hierarchy within the directory named <i>controlpoint</i> accessible to the current environment. Specify the fully qualified, in other words, absolute pathname for the directory. |
| <i>help</i>             | List help information for <i>nsecontrolpt</i> .                                                                                                                                                                          |
| <i>list</i>             | List the control points for the current environment.                                                                                                                                                                     |
| <i>quit</i>             | Exit interactive session.                                                                                                                                                                                                |
| <i>set env</i>          | Use environment <i>env</i> as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default environment using the <i>set</i> command.    |

nseenv

nseenv [-e *env*] [-i] *command arg ...*

The `nseenv` command manipulates NSE environments.

When the user specifies an environment argument to `nseenv`, `nseenv` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes `nseenv` within an activated environment, `nseenv` assumes that environment is the focus of the `nseenv` command, unless the user changes the environment focus using the `-e` option of the `nseenv` command, or using the `nseenv set` command in interactive sessions or script files (described below).

When the user invokes `nseenv` outside of an activated environment, the user must specify the environment focus with the `-e` option or an error may result.

Options:            `-e env`            Set environment in which to execute commands to *env*.

`-i`            Interactive session.

Commands:        Users can specify these commands in one of these ways:

- 1) On the `nseenv` command line.
- 2) In a file of commands, in other words, a script file, provided as standard input to `nseenv`. For example, the user can specify a script file called `scriptfile` to the `nseenv` command by typing:

```
nseenv < scriptfile
```

- 3) In an interactive session. The `-i` option to `nseenv` starts an interactive session.

`children [-s]`    List the names of the child environments of the current environment.

Option:

`-s`            List only the names of the child environments that share files with the current environment.

`create [-m machine] [-r root] [-v variant] ... env controlpoint ...`



Create a root environment called *env* and make this environment the focus for subsequent commands in interactive sessions or script files, until the user changes the focus again with the `nseenv create` or `nseenv set` command. The default variant for the new environment is the variant associated with the current machine architecture, the name of which the user can obtain by invoking the `arch` command. For more information on the `arch(1)` command, see the Man Page, online or in the *Commands Reference Manual*.

The user may specify one or more control points, or directories whose files are controlled by the NSE. Most environments have only a few control points. For more information on control points, see the `nsecontrolpt` command, or Section 3.2, *Control Points*, of the *Network Software Environment: Installation and Administration* manual.

Users on machines other than the machine where the owner created the new environment won't be able to access the environment for about one hour. The Yellow Pages takes about one hour to propagate information about the new environment throughout the network.

When invoked by the administrative username `Nse`, the `nseenv create` command creates a public environment. Public environments are not accessible to users other than `Nse` until the yellow pages propagates news of their existence; usually, this

takes about an hour after creation of the public environment.

When creating an environment as a child environment of some parent environment, use the `acquire` command with the `-c` option, instead of the `nseenv` command.

The `-e` option to `nseenv` has no effect on `nseenv create`.

#### Options:

`-m machine` Create environment on **machine** *machine*; the default is the machine the user invoked the `nseenv` command on.

`-r root` Create the environment using disk space in a directory under the **root** directory named *root*. Because the NSE stores the environment in a directory *under* the root directory, the user can reuse the same root directory name for more than one environment's root directory. By default, the NSE stores environments in the directory `~/.nse_data`.

When the user wants users on other machines to be able to activate the environment, the user must locate the environment on an exported file system directory, particularly when the root directory is on the local disk of a user's machine.

`-v variant` Create the environment with the variant named *variant*. The user may specify more than one variant argument with multiple `-v` options. The default variant is the variant of the machine on which the owner of the environment creates the environment.

`delete [-f] env ...`

When the user is the owner of the environment, delete the environment



called *env*. The user can't delete the current environment, so the `-e` option to `nseenv` has no effect on `nseenv delete`.

The user must own the environment to be able to delete it. However, the administrative username `Nse` can delete public environments, as well as environments owned by any other user.

To deactivate an environment, type CTRL-D to the activated environment command prompt.

The user can't delete an environment until after deactivating the environment.

#### Option:

`-f`

Force deletion of an environment that has children.

`execset execsetname`

Assign the executable set named *execsetname* to the current environment. The executable set appears only in environments activated after making the executable set assignment. For information about how to construct the executable set, see the `nseexecset` command. See also the `nseenv rmexecset` command. For additional information about executable sets, see Section 3.6, *Executable Sets*, in the *Network Software Environment: Installation and Administration* manual.

`help`

List help information for `nseenv`.

`history [-d] [-l]`

List all the revisions of the current environment, including the global revision number, the revision name, the creator of the revision, the time of

the revision, and the components preserved for each revision.

Options:

- d Listing of disk usage for each variant in each revision of the current environment.
- l Long listing, including the comments, of the revision information for the current environment. Same output format as the `nsecomp history -l` command.

`info` List the name, owner, machine, root directory, control points, and variants for the current environment.

`list [-o owner] [-p]` List environments owned by the user who invokes the command.

Options:

- o *owner* List environments owned by user *owner*. This option provides an error when the user attempts to list environments owned by users whose home directories are not mounted on the machine where the user invokes the command.
- p List the public environments. Public environments are not accessible to users other than their creator, the administrative user Nse, until the yellow pages propagates news of their existence; usually, this takes about an hour after creation of the public environment. Same as the `nseenv list -o Nse` command.

`lock` Obtain an environment lock for the current username on the current environment so that only the current user can activate the environment for read and write access. When another



user already has a lock on the environment, `nseenv lock` doesn't lock the environment, but lists the username of the user that has locked that environment.

When the user makes changes to a locked environment that has children, the user should preserve the changed components before unlocking that environment; otherwise, the user may lose some of the changes.

See the `nseenv unlock` command.

`machine [machine]` With no argument, list the name of the machine that "owns" the current environment. The machine that owns the environment runs daemons that permit access to that environment.

When a user who owns the current environment invokes `nseenv machine` with the optional *machine* argument, the command registers machine *machine* as the owner of the current environment.

The Yellow Pages requires about one hour to propagate information about the new owner of the environment throughout the network.

The administrative username `Nse` is the only user who can change the machine registration of public environments, as well as the registration of environments owned by any other user.

`moverev variant comp [rev] dir`

or

`moverev -R`

Move the revision *rev* of the component *comp* and variant *variant* of the current environment so as to store *rev* in an alternate root directory *dir*.

nseenv moverev doesn't permit moving the revisions of an activated environment.

Only the owner of an environment, or the administrative user Nse can move the revisions of that environment.

The nseenv moverev command permits an NSE administrator to free up disk space used for storing a revision in one part of a file system by relocating the revision in another location on the file system resident on another disk drive. The administrative username Nse can move revisions for environments owned by any user.

For example, an administrator can create an environment called env1 with a root directory of /usr/public/envs and a control point of /usr/src, using the following command:

```
nseenv create -r /usr/public/envs env1 /usr/src
```

The revisions of a component comp1 in environment env1 may grow to take up more disk space than is available on the disk drive for /usr/public/envs. In particular, when comp1 [rev1] is taking up a lot of disk space, the administrator can create another directory called /usr/altpublic on another disk drive.

Then, the administrator executes the nseenv moverev command to relocate the storage for comp1 [rev1] into the storage under /usr/altpublic with the command:

```
nseenv -e env1 moverev sun2 comp1[rev1] /usr/altpublic
```



The `comp1[rev1]` revision of environment `env1` is accessible in the same way as it was before, but the administrator has moved the revision to manage the disk space on the system.

Whenever a user moves the revisions of an environment, the user should make sure that to deactivate all environments activated with that executable set, as well as killing the `tf sd` daemons on all the machines that own these activated environments. To kill the `tf sd` daemons:

Log in to the appropriate machine, switch users, using the `su(1)` command, to user `root`. Invoke the `ps(1)` command:

```
ps ax | grep tf sd
```

Invoke the `kill(1)` command on the process ids for any `tf sd` daemons listed in the output of the `ps` command.

Option:

-R

Recover from, in other words, attempt to complete, but otherwise undo an unsuccessful `nseenv moverev` command.

`movevar variant dir`

or

`movevar -R`

Move the variant *variant* of the current environment so as to store *variant* in an alternate root directory *dir*.

`nseenv movevar` doesn't permit moving the variants of an activated environment.

The `nseenv movevar` command permits an NSE administrator to free

up disk space used for storing a variant in one part of a file system by relocating the variant in another location on the file system resident on another disk drive. The administrative username Nse can move variants for environments owned by any user.

For example, an administrator can create an environment called `env1` with a root directory of `/usr/public/envs` and a control point of `/usr/src`, using the following command:

```
nseenv create -r /usr/public/envs env1 /usr/src
```

The variants of environment `env1` may grow to take up more disk space than is available on the disk drive for `/usr/public/envs`. At this point, the administrator can create a directory called `/usr/variants` on another disk drive.

Then, the administrator executes the `nseenv movevar` command to relocate the `sun2` variant of environment `env1` into storage `/usr/variants`:

```
nseenv -e env1 movevar sun2 /usr/variants
```

The `sun2` variant of environment `env1` is accessible in the same way as it was before, but the administrator has moved the variant to manage the disk space on the system.

Whenever a user moves the variants of an environment, the user should make sure that to deactivate all environments activated with that executable set, as well as killing the `tfsd` daemons on all the machines that own these activated



environments. To kill the `tfds` daemons:

Log in to the appropriate machine, switch users, using the `su(1)` command, to user `root`. Invoke the `ps(1)` command:

```
ps ax | grep tfds
```

Invoke the `kill(1)` command on the process ids for any `tfds` daemons listed in the output of the `ps` command.

Option:

`-R`

Recover from, in other words, attempt to complete, but otherwise undo an unsuccessful `nseenv movevar` command.

`name [env]`

Without the `env` argument, list the name of the current environment, if any.

The command changes the name of the current environment to `env` when a user who owns the current environment invokes the `nseenv name` command with the `env` argument.

The administrative username `Nse` is the only user who can change the name of public environments, as well as environments owned by any other user.

`owner user[@machine]`

When invoked by the administrative username `Nse`, change the owner of the current private environment to `user`, and optionally, register the environment on machine `machine`. When invoked by any other user, do nothing

`parents`

List the name of the parent environment of the current environment.

`private user[@machine]`

Make the current public environment a private environment owned by user *user*, and optionally, registered on machine *machine*. Newly private environments are not accessible to users other than Nse until the yellow pages propagates news of their existence; usually, this takes about an hour after creation of the newly private environment. Only takes effect when invoked by the administrative username Nse.

`public`

Make the current environment a public environment. Public environments are not accessible to users other than Nse until the yellow pages propagates news of their existence; usually, this takes about an hour after creation of the public environment. Only takes effect when invoked by the administrative username Nse.

`quit`

Exit interactive session.

`rmexecset execsetname`

Remove the executable set called *execsetname* from the current environment. Doesn't delete the executable set. See also the `nseenv execset` command. For additional information about executable sets, see Section 3.6, *Executable Sets*, in the *Network Software Environment: Installation and Administration* manual.

`rootdir`

List the name of the root directory of the current environment.

`set env`

Use environment *env* as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default



environment using the `nseenv`  
`create` or `nseenv set` command.

`sizerev comp[rev] ...`

List the disk usage for each variant of *comp* in *rev*. When typing to a SunOS shell prompt, the user should preface each square bracket with a backslash character (\) so that the shell interprets the square bracket as a literal square bracket.

`unlock`

Release the environment lock on the current environment that the current user obtained using the `nseenv lock` command. The administrative username `Nse` can release the lock on environments owned by any user.

When the user makes changes to a locked environment that has children, the user should preserve the changed components before unlocking that environment; otherwise, the user may lose some of the changes.

See the `nseenv lock` command.

nseexecset

```
nseexecset [-e execsetname] [-h host_arch host_rel]
          [-i] [-t targ_arch targ_rel] command arg ...
```

The `nseexecset` command manipulates NSE executable sets. The `-h` and `-t` options for `nseexecset` commands are required for all commands that have those options so as to identify the executable set desired by the user. However, the user need not specify the `-h` option when the user invokes an `nseexecset` command on a machine with the same architecture and release as the desired or existing host architecture and release of the executable set. Likewise, the user need not specify the `-t` option when the user invokes the command on a machine with the same architecture and release as the desired or existing target architecture and release of the executable set.

Whenever a user makes a change to an executable set, the user should make sure that to deactivate all environments activated with that executable set, as well as killing the `tfstd` daemon on all the machines that own these activated environments. To kill the `tfstd` daemon:

- 1) Switch users, using the `su(1)` command, to user `root`.
- 2) Invoke the `ps(1)` command:

```
ps ax | grep tfstd
```

- 3) Invoke the `kill` command on the process ids for any `tfstd` daemons listed in the output of the `ps` command.

## Options:

- `-e` Set executable set in which to execute commands to *execsetname*.
- `-h host_arch host_rel` Specify host architecture and release of executable set.
- `-i` Interactive session.
- `-t targ_arch targ_rel` Specify target architecture and release of executable set.

## Commands:

Users can specify these commands in one of these ways:

- 1) On the `nseexecset` command line.
- 2) In a file of commands, in other words, a script file, provided as standard input to `nseexecset`. For



example, the user can specify a script file called *scriptfile* to the *nseexecset* command by typing:

```
nseexecset < scriptfile
```

- 3) In an interactive session. The *-i* option to *nseexecset* starts an interactive session.

*adddir [-o] dir* Add directory named *dir* to the list of directories where executable set items appear. By default, the *nseexecset adddir* command adds a “translucent” directory; in other words, the files located in the user’s native file system appear within the corresponding executable set directories.

Option:

*-o* Add an “opaque” directory, instead of a “translucent” directory; in other words, add a directory in which only the files that the user adds to the executable set appear.

*create [-h host\_arch host\_rel] [-t targ\_arch targ\_rel] execsetname root file*

Create an executable set called *execsetname* for host architecture *host\_arch*, host release *host\_rel*, target architecture *targ\_arch*, and target release *targ\_rel*, with a root directory pathname of *root* for storage of the executable set and a file called *file* that contain a list of directories where executable set items appear. Make this executable set the focus for subsequent commands in interactive sessions or script files, until the user changes the focus again with the *nseexecset create* or *nseexecset set* command.

The file *file* contains a list of directories where executable set items appear, one per line, with a tab or

space characters, and the type of directory, O for opaque or T for translucent, listed after each directory name . Specify the fully qualified, in other words, absolute pathname of the file.

The default host architecture is the architecture of the machine where the user invokes the command. The default host release is the operating system release of the machine where the user invokes the command.

The default target architecture is the architecture of the machine where the user invokes the command. The default target release is the operating system release of the machine where the user invokes the command.

Executable sets are not accessible to users until the yellow pages propagates news of their existence; usually, this takes about an hour after creation of the executable set.

#### Options:

-h Specify the **host** architecture and release of the executable set.

-t Specify the **target** architecture and release of the executable set.

deldir *dir* For the current executable set, delete directory named *dir* from the list of directories where executable set items appear.

delete Delete the current executable set.

dirtytype *dir type* For the current executable set, set the directory named *dir*, from the list of directories where executable set items appear, to type *type*. *type* is either opaque, represented by O, or translucent, represented by T. A translucent directory permits the



|                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                      | contents of the directory in the user's native file system to appear in the directory; whereas, the user must specifically add any files to an opaque directory so that the files appear in that executable set directory.                                                                                                                                                     |
| help                                                                                                 | List help information for the nseexecset command.                                                                                                                                                                                                                                                                                                                              |
| info [-l]                                                                                            | List the host architecture, host release, target architecture, target release, and all of the directories contained in the current executable set.                                                                                                                                                                                                                             |
| Option:                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                |
| -l                                                                                                   | Long listing, including list of all of the files contained in the current executable set.                                                                                                                                                                                                                                                                                      |
| list                                                                                                 | List all of the executable sets in the yellow pages domain of the user who invokes the command.                                                                                                                                                                                                                                                                                |
| prefix                                                                                               | Provide the pathname of the directory storing the current executable set, so that the user can access the files in that executable set.<br><br>Using the prefix pathname <i>prefixpath</i> provided as output from the nseexecset prefix command, the user can copy a file into an executable set:<br><br><div style="text-align: right;"><i>cp file prefixpath/file</i></div> |
| quit                                                                                                 | Exit interactive session.                                                                                                                                                                                                                                                                                                                                                      |
| set [-h <i>host_arch</i> <i>host_rel</i> ] [-t <i>targ_arch</i> <i>targ_rel</i> ] <i>execsetname</i> | Use executable set <i>execsetname</i> as default executable set for commands in interactive session or script file, until user either quits the interactive session or resets the default                                                                                                                                                                                      |

executable set using the  
nseexecset create or  
nseexecset set command.

Options:

- h Specify the **host** architecture and  
release of the executable set.
- t Specify the **target** architecture and  
release of the executable set.



**nsegc****nsegc [-n] *env numberofdays***

The **nsegc** command removes files that are no longer needed from environment *env*. Only the owner of an environment, or the administrative user **Nse** can clean up these files using **nsegc**, the NSE “garbage collection” command. The reason to remove these unneeded environment files is to conserve disk space.

**nsegc** removes all of the files from the environment *env* that no user has accessed within the past *numberofdays*, subject to the following constraints:

- **nsegc** doesn’t remove any files that the environment *env* shares with any other environment.
- **nsegc** doesn’t remove any files from named revisions. This is because a user may at any time wish to acquire an old named revision.
- **nsegc** doesn’t remove the latest revision of any file. Similar to the situation for old named revisions, a user may acquire or resync with the latest revision of a file.

Before **nsegc** can garbage collect an environment, the user must invoke the **nsemark** command on *env* and on each environment that shared files with *env* within the past *numberofdays*. To find out which environments shared files with *env*, see the **nseenv children -s** command. **nsegc** automatically verifies that a user has invoked **nsemark** on all of the necessary environments. When **nsegc** finds an environment that the user still needs to mark using **nsemark**, **nsegc** provides an error message which includes a list of the unmarked environments.

For example, when user `tutorial` runs `nsecg` on environment `env1` that has, over the past three days, shared files with environments `env2` and `env3`, owned by user `tutorial`, and environment `env4`, owned by user `brick` on machine `mendacity`, after already marking environment `env2` using `nsemark`, and not marked environment `env4` over the past three days, the following listing could result:

Figure 4-23 *Garbage Collection of Unmarked Environments*

```
tutorial% date
Sat Oct 31 12:00:00 PST 1987
tutorial% nsecg env1 3
nsecg: garbage collection cannot proceed -- the following environments
have not been marked within the past 3 days:
    env3 - never marked.
    env4@brick@mendacity - 10/26/87 00:00:07
tutorial%
```

For more information on garbage collection, see Section 3.8, *Removing Unneeded Files from Revisions*, in the *Network Software Environment: Installation and Administration* manual.

|         |    |                                                                                                        |
|---------|----|--------------------------------------------------------------------------------------------------------|
| Option: | -n | Do not garbage collect. List any environments user needs to mark before performing garbage collection. |
|---------|----|--------------------------------------------------------------------------------------------------------|



**nseivp****nseivp**

**nseivp** is an administrative script designed to verify a number of aspects of the NSE installation. When the user has trouble activating environments or running NSE executables, the user should run the script on the user's machine and on any machine where a troublesome environments resides.

The tests performed by the **nseivp** command are not exhaustive, but check a wide variety of potential problem areas. The **nseivp** command signals successful completion with the message:

NSE installation successfully verified.

However, **nseivp** can't check all possible problems, so even when **nseivp** signals successful completion, it doesn't guarantee that the installation is correct.

Unlike most of the commands, which are located in `/usr/nse/bin`, **nseivp** is located in the `/usr/nse/etc` directory.

For more information on **nseivp**, see the *Network Software Environment: Installation and Administration* manual.

## nsemark

### nsemark *env*

The `nsemark` command marks an environment *env* so that the `nsegc` command can “garbage collect” unneeded files from that environment and from its parent environment.

To the NSE, an unneeded file is any file that is:

- In an unnamed revision, and
- Not being shared by another environment, and
- Not part of the latest revision of a component.

The user must run the `nsemark` command on all environments that share with the environment the user wants to garbage collect using the `nsegc` command. To find out which environments shared files with *env*, see the `nseenv children -s` command.

For more information on marking environments for garbage collection, see Section 3.8, *Removing Unneeded Files from Revisions*, in the *Network Software Environment: Installation and Administration* manual.



nsenotify

nsenotify [-e *env*] [-i] *command arg ...*

The `nsenotify` command is the command-line interface to the NSE's notification facility. The notification facility permits the user to make *requests* to detect the occurrence of certain *events* and to schedule *actions* that occur in response to those events.

The `nsenotify` command can detect any of these events:

Table 4-1 Notification Events

| <i>Event</i> | <i>Description</i>                          |
|--------------|---------------------------------------------|
| acquire-from | A user acquires from the environment        |
| resync-from  | A user resyncs from the environment         |
| reconcile-to | A user reconciles to the environment        |
| envlock      | A user places a lock on the environment     |
| envunlock    | A user releases a lock from the environment |

An action is either sending a mail message or executing a program specified by the user who makes the request.

The `nsenotify` command permits the user to register, unregister, and list notification requests. Only the user who registers the notification request, or the administrative user `Nse`, may remove a request.

Each event from the list above has event-specific arguments, such as *type* and *object*, associated with that event:

Table 4-2 Notification Event-Specific Arguments

| <i>Event</i> | <i>Event-Specific Arguments</i> |
|--------------|---------------------------------|
| acquire-from | <i>type object</i>              |
| resync-from  | <i>type object</i>              |
| reconcile-to | <i>type object</i>              |
| envlock      | None                            |
| envunlock    | None                            |

*type* is the type of object, and *object* the name of the object, on which the user believes the event will take place.

When a notification event completes, the notification facility reports type-specific information about the event. In most cases, the notification facility simply reports the name of the object. However, in two cases, the `reconcile-to` event for an object of type `Comp` and the `reconcile-to` event for an

object of type *Targ*, the notification facility reports may contain other objects within the specified component or target that also were reconciled.

When the user specifies a file or link database name for *object*, the user must provide the fully qualified, in other words, absolute pathname for the file or link database.

When the user specifies a target, the user must provide the target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname of the Makefile associated with the target.

The syntax for component names consists of a root component, specified by a colon (:), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

The user may not specify a revision name for any *component* arguments of the *nsenotify* command.

|          |                                                                                                                                                                                                                                                                                        |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Options: | <div style="display: flex; justify-content: space-between;"> <div style="width: 40%;"> <p><code>-e env</code></p> <p><code>-i</code></p> </div> <div style="width: 60%;"> <p>Set environment in which to execute commands to <i>env</i>.</p> <p>Interactive session.</p> </div> </div> |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|           |                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Commands: | <p>Users can specify these commands in one of these ways:</p> <ol style="list-style-type: none"> <li>1) On the <i>nsenotify</i> command line.</li> <li>2) In a file of commands, in other words, a script file, provided as standard input to <i>nsenotify</i>. For example, the user can specify a script file called <i>scriptfile</i> to the <i>nsenotify</i> command by typing:</li> </ol> |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
nsenotify < scriptfile
```

- 3) In an interactive session. The `-i` option to *nsenotify* starts an interactive session.

|      |                                                         |
|------|---------------------------------------------------------|
| help | List help information for the <i>nsenotify</i> command. |
|------|---------------------------------------------------------|

|                              |                                                                                                                                                                                   |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| list [-a] [-v <i>event</i> ] | <p>List all notification requests for the user who invokes the <i>nsenotify</i> <i>list</i> command. This command provides a request id for each notification request listed.</p> |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



## Options:

- a List notification requests for all users who have notifications requests pending for the current environment.
- v *event* List all notification requests for event *event* pending for the current environment.
- quit Exit interactive session.
- register [-c *command arg ...*] [-m *machine*] [-u *username*] *event event-specific-arg ...*

Register a notification request to trigger upon detection of event *event*. When the user doesn't specify any options to the `nsenotify register` command, the command mails an electronic message to the user and the machine that invoked the `nsenotify` command.

## Options:

- c *command arg ...*

Execute program called *command* and associated arguments on the current machine, or the machine specified with the `-m` option, upon detection of the event. The mail daemon executes *command*, unless *command* uses the `setuid(3)` routine to run as another user. There are six predefined variables that the user may pass as arguments to *command*. They are:

- user* The name of the user who executed the command that triggered the event.
- env* The environment where the event took place.
- event* One of the events described above.
- object* The name of the object on which the event occurred.

- type*            The type of the object in which the event occurred.
- typeinfo*       A filename that contains details about execution of the notification event.
- `-m machine`    Mail to machine named *machine*. Default is machine where user invoked `nsenotify register` command.
- `-u username`   Mail to user named *username*. Default is user who invoked `nsenotify register` command.
- `set env`        Use environment *env* as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default environment using the `nsenotify set` command.
- `unregister event requestid`
- Unregister notification request number *requestid* that would have triggered upon detection of event *event*. To obtain the *requestid* of a notification request, use the `nsenotify list` command.



**nsesymlink****nsesymlink** [-e *env*] [-i] *command arg ...*

The **nsesymlink** command manipulates the set of symbolic links associated with an NSE environment. Symbolic links permit users to refer to the same file with two different pathnames.

The **nsesymlink** command can only update the symbolic links of a root environment, in other words, an environment that has no parent environment. While in an interactive session (described below), **nsesymlink** prompts for confirmation before attempting to update the symbolic links of an environment that is not a root environment.

Once a user updates the symbolic links in a root environment, users can acquire the symbolic links into the child environments of the root environment. Administrators can use **nsesymlink acquire** to propagate changes to symbolic links throughout the environment hierarchy of the root environment.

Changes to symbolic links in the root environment also propagate automatically throughout the environment hierarchy when the user resyncs child environments with the root environment, then resyncs the child environments of those children, and so on.

The user must not change the symbolic links associated with a child environment if the user ever wants to be able to reconcile that child environment with its parent environment.

When the user specifies an environment argument to **nsesymlink**, **nsesymlink** returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes **nsesymlink** within an activated environment, **nsesymlink** assumes that environment is the focus of the **nsesymlink** command, unless the user changes the environment focus using the **-e** option to the **nsesymlink** command, or using the **nsesymlink set** command in interactive sessions or script files (described below).

When the user invokes **nsesymlink** outside of an activated environment, the user must specify the environment focus with the **-e** option or an error may result.

For more information on symbolic links, see the **ln(1)** Man Page, online or in the *Commands Reference Manual*.

|          |               |                                                              |
|----------|---------------|--------------------------------------------------------------|
| Options: | -e <i>env</i> | Set environment in which to execute commands to <i>env</i> . |
|          | -i            | Interactive session.                                         |

Commands: Users can specify these commands in one of these ways:

- 1) On the `nsesymlink` command line.
- 2) In a file of commands, in other words, a script file, provided as standard input to `nsesymlink`. For example, the user can specify a script file called `scriptfile` to the `nsesymlink` command by typing:

```
nsesymlink < scriptfile
```

- 3) In an interactive session. The `-i` option to `nsesymlink` starts an interactive session.

`acquire parentenv` Acquire the symbolic links from *parentenv* to the current environment. Use `nsesymlink acquire` when the current environment has a parent environment that contains the desired symbolic links.

A user can propagate changes to symbolic links throughout the environment hierarchy of the root environment with repeated use of the `nsesymlink acquire` command.

In most cases, the user acquires the symbolic links from the parent environment of the current environment using the `acquire` command with the `-c` option, or using `nsesymlink acquire`.

`add` Initialize the current environment so that the user may associate symbolic links with that environment. Use `nsesymlink add` only with a root environment, in other words, when the current environment has no parent environment.

In most cases, the user acquires the symbolic links from the parent



environment of the current environment using the `acquire` command with the `-c` option, or using `nsesymlink acquire`.

`append source dest` Add a new symbolic link called *source* that points at the file named *dest* in the current environment. Don't use this command; use `nsesymlink assign` instead. The `bootstrap` command produces script files containing this command during the bootstrap process.

`assign source [dest]`

With one argument, `assign` deletes the symbolic link named *source* in the current environment. With two arguments, `assign` a symbolic link called *source* point at the file named *dest* in the current environment. Specify the fully qualified, in other words, absolute pathnames for *source* and *dest*.

To manipulate symbolic links within the NSE, use the `nsesymlink assign` command instead of the `ln -s` or `rm` commands

`help`

List help information for `nsesymlink`.

`list`

List the symbolic links table for the current environment.

`quit`

Exit interactive session.

`set env`

Use environment *env* as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default environment using the `set` command.

nsevar

nsevar [-e *env*] [-i] *command arg ...*

The `nsevar` command manipulates the variants of NSE environments. Variants are a facility that permits users to produce executable files for a variety of architectures from a shared set of sources.

The `nsevar` command can only update the variants of a root environment, in other words, an environment that has no parent environment.

Once a user updates the variants in a root environment, users can acquire these variants into the child environments of the root environment. Administrators can use `nsevar acquire` to propagate changes to variants throughout the environment hierarchy of the root environment.

Changes to variants in the root environment also propagate automatically throughout the environment hierarchy when the user resyncs child environments with the root environment, then resyncs the child environments of those children, and so on.

The user must not change the variants associated with a child environment if the user ever wants to be able to reconcile that child environment with its parent environment. While in an interactive session (described below), `nsevar` prompts for confirmation before attempting to update the variants of an environment that is not a root environment.

When the user specifies an environment argument to `nsevar`, `nsevar` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes `nsevar` within an activated environment, `nsevar` assumes that environment is the focus of the `nsevar` command, unless the user changes the environment focus using the `-e` option to the `nsevar` command, or using the `nsevar set` command in interactive sessions or script files (described below).

When the user invokes `nsevar` outside of an activated environment, the user must specify the environment focus with the `-e` option or an error may result.

**Note:** Sometimes the `reconcile` command incorporates a `resync`, which propagates the variant changes to children.

Options:

`-e env`

Set environment in which to execute commands to *env*.

`-i`

Interactive session.



## Commands:

Users can specify these commands in one of these ways:

- 1) On the `nsevar` command line.
- 2) In a file of commands, in other words, a script file, provided as standard input to `nsevar`. For example, the user can specify a script file called `scriptfile` to the `nsevar` command by typing:

```
nsevar < scriptfile
```

- 3) In an interactive session. The `-i` option to `nsevar` starts an interactive session.

`acquire` *parentenv* *variant*

Acquire a variant called *variant* from *parentenv* to the current environment. Use `nsevar acquire` when the current environment has a parent environment that contains the desired variant.

Only the owner of the environment, or the administrative user `Nse` can acquire a variant into an environment.

The administrative user `Nse` can propagate changes to variants throughout the environment hierarchy of the root environment with repeated use of the `nsevar acquire` command. See the description of the `nseenv children` command to find out how to list the child environments of an environment in that environment hierarchy.

When the user needs to update a child environment with an additional variant from its parent environment, the user should invoke the `nsevar acquire` command, rather than trying to add a new variant using the `nsevar add` command. Adding a variant with the `nsevar add` command prevents the user from

*add variant*

reconciling from that child environment to its parent environment.

Add a variant called *variant*. Use `nsevar add` only with a root environment, in other words only when the current environment has no parent environment because adding a variant with the `nsevar add` command prevents the user from reconciling from that child environment to its parent environment.

Only the owner of the environment, or the administrative user Nse can add a variant to an environment.

In most cases, the user acquires the variant from the parent environment of the current environment using `acquire` with the `-c` and `-v` options, or using `nsevar acquire`.

*assign variant shellvar [value]*

Assign the shell environment variable *shellvar* associated with the variant specification for the variant named *variant* to the value *value*. When the user doesn't specify the optional *value* argument, `assign` removes *shellvar* from the list of variables associated with the variant *variant*. Activating an environment assigns the *value* values to the *shellvar* shell environment variables for the duration of that activation.

The user should only assign values to variables for variants in a root environment, in other words, an environment with no parent environment.

Only the owner of the environment, or the administrative user Nse can assign values to variant variables in an environment.



|                                    |                                                                                                                                                                                                                             |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | The administrative user Nse can propagate changes to variants throughout the environment hierarchy of the root environment with repeated use of the <code>nsevar acquire</code> command.                                    |
| <code>delete <i>variant</i></code> | Delete variant called <i>variant</i> from the current environment.<br><br>Only the owner of the environment, or the administrative user Nse can delete a variant from an environment.                                       |
| <code>help</code>                  | List help information for <code>nsevar</code> .                                                                                                                                                                             |
| <code>info <i>variant</i></code>   | List the values for the shell environment variables in the variant specification for <i>variant</i> . See the <code>nsevar assign</code> command.                                                                           |
| <code>list</code>                  | List variants associated with the current environment.                                                                                                                                                                      |
| <code>quit</code>                  | Exit interactive session.                                                                                                                                                                                                   |
| <code>set <i>env</i></code>        | Use environment <i>env</i> as default environment for commands in interactive session or script file, until user either quits the interactive session or resets the default environment using the <code>set</code> command. |

preserve

```
preserve [-c comment] [-f] [-q] [-r revname] [-t]  
[-Q] component...
```

or

```
preserve -C [-q] component ...
```

or

```
preserve -R [-q]
```

The `preserve` command makes a revision of an NSE component or components within the currently activated environment.

`preserve` asks for a comment interactively when the user doesn't specify the comment using the `-c` option.

By default, the `preserve` command makes a partial base revision. A partial base revision makes and saves copies only of those files that have changed since the last revision within the specified component or components. In most cases, individual users of the NSE should make partial base revisions.

When the user specifies the `-t` option, the `preserve` command creates a total base revision. A total base revision makes and saves copies of all of the files within the specified component or components. An administrator should create total base revisions to speed up access to files in the environment. Create total base revisions only on root components.

The user usually specifies the `-r` option to name the revision created by `preserve`. Named revisions are permanent, and therefore retrievable. The user can't retrieve unnamed revisions; they are useful only as the latest consistent revision for certain commands. Whenever the user makes another revision of a component, the unnamed revision of the component is no longer accessible.

The syntax for component names consists of a root component, specified by a colon (:), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

The user may not specify a revision name for any *component* arguments to the `preserve` command.



|          |                                |                                                                                                                                                                                                                                                                                                                                      |
|----------|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Options: | <code>-c <i>comment</i></code> | Specify <b>comment</b> associated with new revision. Enclose <i>comment</i> in quotes when there are space characters in it. <code>preserve</code> prompts for the comment when the user doesn't specify it using the <code>-c</code> option.                                                                                        |
|          | <code>-f</code>                | Force a <code>preserve</code> of all subobjects of the components specified even if the subobjects haven't changed.                                                                                                                                                                                                                  |
|          | <code>-q</code>                | Quiet execution prevents command from printing out any messages, except error messages.                                                                                                                                                                                                                                              |
|          | <code>-r <i>revname</i></code> | Name revision created as <i>revname</i> . <i>revname</i> must not contain any space characters and must not be longer than 14 characters.                                                                                                                                                                                            |
|          | <code>-t</code>                | Total base – make new revision a total base, saving copies of all files in the specified component or components. <b>Do not</b> use the <code>-t</code> option on any component other than the root component, represented by a colon (:), of the currently activated environment, or you may inadvertently destroy some files.      |
|          | <code>-C</code>                | Don't actually do the <code>preserve</code> , just list any unstable objects in the component, for example, list any files that someone still has checked out with <code>vcs</code> .                                                                                                                                                |
|          | <code>-Q</code>                | Quick revision. Make revision of only the component or components the user specifies in the command. Make no revisions of subobjects in the component or components the user specifies. Useful when the user changes a component, in other words, adds a new subcomponent, but the remaining subobjects have already been preserved. |

-R

**Recover from, in other words undo, an interrupted preserve.**



reconcile

reconcile [-c] [-n] [-p *parentenv*] [-q] [-r *revname*]  
[-N] *component* ...

or

reconcile -R [-p *parentenv*] [-q]

`reconcile` updates an NSE parent environment component or components with changes in the corresponding component or components from the child environment. When `reconcile` can't update the parent environment because users have changed objects in both the parent and child environments, `reconcile` automatically invokes `resync` to acquire the parent environment's versions of the objects. Then, the user can merge the parent environment versions of the objects with the child environment versions using the `resolve` command. When the specified component or components in the child environment are identical to the corresponding component or components in the parent environment, `reconcile` doesn't do anything.

When the specified component or components in the parent environment have not changed since the user acquired or reconciled the component or components into the child environment, `reconcile` automatically updates the parent environment by substituting the changes made in the child component or components into the corresponding component or components of the parent environment.

When someone has changed the component or components in the parent environment since the user last acquired or reconciled the corresponding component or components into the child environment, `reconcile` lists the conflicts between the contents of the component or components in the child environment and in the parent environment. Then, `reconcile` invokes the `resync` command. The user must then invoke the `resolve` command to resolve the child environment with its parent environment so that the appropriate changes make it into the object hierarchy of the parent environment.

When the user wants to pass options to the `resync` command, the user can `reconcile` without automatically resyncing by invoking the `reconcile` command with the `-N` option.

When the user invokes `reconcile` on a component or components within an activated child environment that has only

one parent environment, `reconcile` can determine the parent environment automatically. However, when the user invokes `reconcile` on a new component within an activated child environment that does not exist in the parent environment, the user must specify, using the `-p` option, the parent environment *parentenv* with which to reconcile.

The `reconcile` command may reconcile components other than those the user specifies as arguments because constrained files may require the involvement of other components. For example, when the user reconciles a component that contains the derived file `libnse.a`, the `reconcile` command requires a reconcile of the component that contains the source file for `libnse.a`, as well as the component that contains `libnse.a`. At the end of each reconcile, `reconcile` makes a revision of the component or components in the parent environment. `reconcile` makes a partial base revision, saving only the files that have changed since the last revision of the specified component or components.

When one user is trying to reconcile objects in a component of a child environment to its parent environment, another user may be making changes to objects within the corresponding component in the parent environment. There are two possibilities at this point:

- 1) The `reconcile` command fails when the user working in the parent environment has some objects checked out, and the user working in the child environment must recover, in other words, undo the `reconcile` command using the `-R` option. The user working in the child environment must persuade the user working in the parent environment to check in all objects that user has checked out and make a revision of the component that contains these objects. Then, the user working in the child environment can once again attempt to reconcile to the parent environment.
- 2) The `reconcile` command issues warnings when the user working in the parent environment has already checked in any changes that user made to objects in the parent environment, but has not made a revision of these changes. `reconcile` copies into the parent environment a new version of each of the objects the user changed in the child environment. If the user working in the child environment, or the user working in



the parent environment wants to retain any of the changes checked in by the person working in the parent environment, the user must merge the changes made in objects in the components of the child environment with the changes made to objects in corresponding components in the parent environment.

Sometimes it is necessary to recover from an aborted `reconcile` command. The user may specify the `-R` option for a reconcile recovery. When more than one recovery is possible, the user must specify, using the `-p` option, the parent environment *parentenv* with which to recover the reconcile.

The syntax for component names consists of a root component, specified by a colon (`:`), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

The user may not specify a revision name for any *component* arguments of the `reconcile` command.

When the user specifies the abbreviation colon (`:`) as the *component* argument to `reconcile`, the command reconciles all of the components in the child environment to the parent environment.

When the user specifies an environment argument to `reconcile`, `reconcile` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes `reconcile` without any arguments, `reconcile` lists usage information for the command.

Options:

`-c`

Reconcile the latest consistent revision of the component in the child environment with the corresponding component in the parent environment; in other words, reconcile the results of the most recent preserve in the child component. The default is to reconcile with the latest version of all the files.

If a user has changed any objects in the child environment component, the `reconcile` fails with an error. If a

- user has made changes to the child environment component, make sure to preserve the component before using the `reconcile -c` command.
- `-n` Do nothing, just list the objects the user has changed in the child environment, and therefore needs to reconcile to the parent environment. The `reconcile -n` command also reports any conflicts between the objects in the child environment and corresponding objects in the parent environment.
- `-p parentenv` Reconcile with the parent environment called *parentenv*.
- `-q` Quiet execution prevents command from printing out any messages, except error messages.
- `-r revname` Specify *revname* of revision in parent created by `reconcile`. *revname* must not contain any space characters and must not be longer than 14 characters.
- `-N` No automatic resync with reconcile.
- `-R` Recover, in other words, undo an interrupted reconcile.

`reconcile -R` doesn't require the user to specify the name of the parent environment with the `-p` option when the user invokes `reconcile` from within an activated environment and there is only one possible parent environment to which a reconcile recovery could occur.

However, when the user invokes `reconcile -R` outside of an activated environment, or within an activated environment with at least one other possible reconcile recovery that could occur, the user must specify



the name of the parent environment using the `-p` option so that `reconcile` can distinguish between multiple possible reconcile recovery operations.

The `-R` option doesn't work with any options except the `-q` option.

## resolve

## resolve [-n]

The `reconcile` command may detect conflicts between the contents of a component or components in a child environment and the corresponding component or components in its parent environment. Either when the `reconcile` command automatically invokes the `resync` command, or when the user manually invokes the `resync` command, the command may detect merge conflicts. A merge conflict occurs when users have modified an object in a child environment and the corresponding object within the component hierarchy of its parent environment. The purpose of the `resolve` command is to merge these conflicts.

The user invokes the `resolve` command to perform the resolution of the child environment with the parent environment. When the user invokes the `resolve` command from a shell or script that doesn't permit execution of window- and mouse-based applications, the `resolve` command lists the objects in the currently activated environment that require merging with corresponding objects in the parent environment. The user then merges the changes made to objects from the child environment with changes made to objects in the parent environment, marks the objects merged, and repeats the attempt to reconcile the component or components in the child environment with the corresponding component or components in the parent environment. For more information, see the command summary for `merged`.

When the user invokes the `resolve` command from a shell capable of causing the display of a window- and mouse-based application, `resolve` invokes the type-specific merge resolution application appropriate for each type of conflict between the contents of a child environment and the contents of its parent environment. When `resolve` encounters an object type for which there is no merge resolution program, `resolve` provides a warning. When `resolve` invokes the window-based applications, the user need not mark the objects that require merging using the `merged` command; the `resolve` command marks these objects automatically.

In the case of resolution of objects of type `File`, `resolve` invokes the `fileresolve` program. `fileresolve` permits the user to resolve sequentially any merge conflicts between files in the child environment component with the corresponding



component of the parent environment. For more information, see the command summary for `fileresolve`.

In the case of resolution of objects of type `Comp` and `Linkdb`, `resolve` invokes the `dbresolve` program. `dbresolve` permits the user to resolve sequentially any merge conflicts between component and link objects in the child environment component with the corresponding component of the parent environment. For more information, see the command summary for `dbresolve`.

|         |                 |                                                                                                                                                                                                                                                                                                 |
|---------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option: | <code>-n</code> | Do nothing, just list the conflicts that the user still needs to resolve between the contents of the component in the child environment and the contents of the component in the parent environment. Also warns of any object types present whose conflicts <code>resolve</code> can't resolve. |
|---------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

resync

```
resync [-m] [-n] [-q] [-x childenv] [-C]  
component ...
```

or

```
resync -R [-q] [-x childenv]
```

resync updates the component or components in the child environment and all the objects that component or components contain with respect to the corresponding component or components in the parent environment.

In most cases, the `reconcile` command invokes `resync` automatically, although a user may, in some circumstances, invoke `resync` separately, or invoke `reconcile` without an automatic `resync`.

By default, `resync` automatically updates objects in the specified component or components of the child environment that have changed in the corresponding component or components of the parent environment if the versions of the objects in the child environment have not changed since the last `acquire`, `resync`, or `reconcile`. However, when developers have changed an object in both the parent and child environments, `resync` causes a "conflict." The user must resolve conflicts before the next `reconcile` from the child environment. Typically, the user resolves conflicts immediately after the `resync` that indicated the conflicts. The user can resolve these conflicts either "by hand," using a text editor or other development tools, or by invoking the `resolve` command.

For file conflicts, `resync` places the conflicting version of the file from the parent environment into the child environment with a filename obtained by adding the suffix `.M` to the name of the conflicting file. Similarly, `resync` places the "common ancestor" file of the conflicting files into the child environment with a filename obtained by adding the suffix `.A` to the name of the conflicting file. The `resolve` command uses these `.A` and `.M` files in resolving file conflicts, or by the user who decides to resolve the files "by hand," merging change conflicts using a text editor. When the user resolves a conflict "by hand," the user indicates completion of a conflict resolution using the `merged` command, so that a later `reconcile` takes note of the user's conflict resolution changes.



The `resync` command provides a warning when it cannot resync a constrained file. This happens when the user attempts to resync a component containing a constrained file without resyncing the component in the same environment that contains the source files associated with the constrained file. `resync` won't resync the constrained file in this case since the constrained file would become inconsistent with the source files. If the user's environment doesn't contain the source files associated with the constrained file, then `resync` can resync the constrained file without producing any inconsistencies.

When a user has added a new object to a compound object in a parent environment, for example, adding a file to a target, the `resync` command involving the compound object, if invoked separately from `reconcile`, won't update the child environment with the new object. To update the child environment with this new object, the user can append the object to the child environment by invoking the `append` command.

The syntax for component names consists of a root component, specified by a colon (`:`), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

When the user specifies an environment argument to `resync`, `resync` returns an error if the user specifies a */variant* suffix of the environment name, since the command applies to all variants of the environment.

When the user invokes `resync` without any arguments, `resync` lists usage information for the command.

|         |                 |                                                                                                                                                                                                                                                                                                                    |
|---------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option: | <code>-m</code> | Don't automatically update objects that have changed in the component or components located in the parent environment, but haven't changed in the component or components located in the child environment. Instead, <b>make</b> a conflict on such objects. The default is to update automatically these objects. |
|---------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- n Do nothing, just list the conflicts that the user still needs to resolve between the contents of the component in the child environment and the contents of the component in the parent environment.
- q Quiet execution prevents command from printing out any messages, except error messages.
- x *childenv* Extend the existing child environment called *childenv*, which is not currently activated. From within an activated child environment, the user invokes *resync* without the -x option to *resync* a component to that child environment.
- C Copy files necessary for *resync* to the current environment rather than trying to share them with *parentenv*.
- R Recover, in other words, cancel an interrupted *resync*.



## VCS

*vcs command arg ...*

*vcs* is the NSE version control system (VCS). VCS provides the user with the ability to keep track of successive versions of files associated with the NSE. Whenever possible, VCS stores only the differences between versions of files.

Commands: *vcs* requires the user to specify one of these commands:

*checkin [-c comment] [-q] file ...*

Check in a version of *file* to save all the changes necessary for the next version. Preserves the source file from which the version is made. Specify the fully qualified, in other words, absolute pathname for the file.

## Options:

- c comment* Specify comment associated with version user checks in. Enclose *comment* in quotes when there are space characters in it. *vcs checkin* prompts for the comment when the user doesn't specify it using the *-c* option.
- q* Quiet execution prevents command from printing out any messages, except error messages.

*checkout [-q] file ...*

Check out the latest version of *file* for use in development, for example, editing. Specify the fully qualified, in other words, absolute pathname for the file.

## Option:

- q* Quiet execution prevents command from printing out any messages, except error messages.

*ci [-c comment] [-q] file ...*

Same as *checkin*.

*co [-q] file ...* Same as *checkout*.

`create [-c comment] [-q] file ...`

Place *file* under VCS control.

Preserves the source file from which the version is made. Specify the fully qualified, in other words, absolute pathname for the file.

Options:

`-c comment` Specify comment associated with version user creates. Enclose *comment* in quotes when there are space characters in it. `vcs create` prompts for the comment when the user doesn't specify it using the `-c` option.

`-q` Quiet execution prevents command from printing out any messages, except error messages.

`free [-c] file ...` Take *file* out from under VCS version control.

Option:

`-c` Treat *file* as a "constrained" file that the user can't rebuild within the current environment because the file is derived elsewhere.

`get [-d date] [-p] [-q] [-v version] file ...`

Retrieve a specified version of the *file*. In contrast to `checkout`, `get` obtains a read-only version of *file*. Specify the fully qualified, in other words, absolute pathname for the file.

Options:

`-d date` Retrieve last version made before *date*. Example date is:

03/03/87 09:06:04

Incompatible with `-v` option.

`-p` Print output to standard output, rather than to file.



|                                 |                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-q</code>                 | Quiet execution prevents command from printing out any messages, except error messages.                                                                                                                                                                                                                                                                                                              |
| <code>-v version</code>         | Retrieve version numbered <i>version</i> .<br>Example version number is:<br>1.3<br>Incompatible with <code>-d</code> option.                                                                                                                                                                                                                                                                         |
| <code>hi file ...</code>        | Same as <code>history</code> .                                                                                                                                                                                                                                                                                                                                                                       |
| <code>history file ...</code>   | Print VCS history associated with <i>file</i> . Specify the fully qualified, in other words, absolute pathname for the file.                                                                                                                                                                                                                                                                         |
| <code>list [directory]</code>   | List any of the files in <i>directory</i> that some user has checked out. Specify the fully qualified, in other words, absolute pathname for the directory.                                                                                                                                                                                                                                          |
| <code>pull file</code>          | Take the version of <i>file</i> shared by all variants associated with the current environment and make it visible only in the current variant of the current environment. Compare with <code>vcs push</code> .                                                                                                                                                                                      |
| <code>push file</code>          | Make the version of <i>file</i> in the current variant visible as the current version of <i>file</i> in all variants associated with the environment name they share. Specify the fully qualified, in other words, absolute pathname for the file. Both <code>vcs checkin</code> and <code>vcs create</code> perform an implicit <code>vcs push</code> command. Compare with <code>vcs pull</code> . |
| <code>rename file1 file2</code> | Change the name of <i>file1</i> to <i>file2</i> . Specify the fully qualified, in other words, absolute pathnames for the files. Use <code>vcs rename</code> instead of the SunOS <code>mv</code> command to rename                                                                                                                                                                                  |

directories and files so that the NSE can properly track the history of the renamed directory or file.

`rmver [-q] -v version file`

Remove the last version of *file*. Must specify the version number of *version* using the `-v` option. Use `vcs uncheckout` to remove the changes made since the previous `vcs checkout`. Specify the fully qualified, in other words, absolute pathname for the file.

Options:

- `-q` Quiet execution prevents command from printing out any messages, except error messages.
- `-v version` Remove the last version numbered *version*. Example version number is:  
1.3

`uncheckout [-q] file ...`

Undo a previous `vcs checkout`, discarding any changes made since that checkout. Use `vcs rmversion` to remove previous versions. Specify the fully qualified, in other words, absolute pathname for the file.

Option:

- `-q` Quiet execution prevents command from printing out any messages, except error messages.

`vdiff [-v version1] [-v version2] file`

When the user doesn't provide any `-v` options, `vdiff` determines the differences between the current working *file* and the last version of *file* that a user checked in. When the user doesn't provide the second `-v` option,



`vdiff` determines the differences between the current working *file* and the specified *version1* of that file. Otherwise, `vdiff` determines the differences between the two specified versions of *file*. Specify the fully qualified, in other words, absolute pathname for the file.

Option:

`-v version`    Number of version with which to determine differences. Example version number is:

1.3

---

## NSE Naming Specifications

|                                 |     |
|---------------------------------|-----|
| NSE Naming Specifications ..... | 253 |
| 5.1 Environment Names.....      | 253 |
| 5.2 Variant Names .....         | 254 |
| 5.3 Component Names.....        | 254 |
| 5.4 Target Names .....          | 255 |
| 5.5 Filenames .....             | 255 |
| 5.6 Link Database Names .....   | 255 |







---

## NSE Naming Specifications

This chapter describes the NSE naming specifications for use in both the window-based Browser and command-line interface to the NSE. For the specifics of the Browser interface, see Chapter 1, *The Browser*. For the specifics of the command-line interface, see Chapter 4, *Command-Line Interface*.

### 5.1 Environment Names

An environment name is a string of alphanumeric characters with a maximum length of 88 characters.

Within the command-line interface, some commands take optional username and machine specifications for environments. These specifications may be necessary to clear up any ambiguity between environments with the same base name. For example, to differentiate environment `env7` owned by user `doe` from environment `env7` owned by user `ray`, a user could specify environment `env7`, owned by user `doe`, and registered on machine `tutorial`, like this:

```
env7@doe@tutorial
```

The environment segment of the environment name has a maximum length of 64 characters. The username segment of the environment name has a maximum length of 8 characters. The machine name segment of the environment name has a maximum length of 16 characters.

In the case of public environments, the owner's username is `Nse`, the NSE administration username, and the commands ignore any machine specification provided by the user. So, a user could specify public environment `pubenv3`, owned by the NSE administration username `Nse`, like this:

```
pubenv3@Nse
```



## 5.2 Variant Names

A variant name is a string of alphanumeric characters that must not cause an environment name to the maximum length of 88 characters.

In the command-line interface, the user may specify an environment name alone, or the environment name with a */variant* suffix. For example, with a command like `activate`, which takes the */variant* suffix to environment arguments, the user can specify environment tools just as `tools`, or the user can specify environment tools with variant Sun-3 like this:

```
tools/sun3
```

**Note:** A standard format for variant names is the output format of the `/bin/arch` command, say `sun3`.

When a user specifies the */variant* suffix to an environment argument of a command that doesn't permit the */variant* suffix specification, the command provides an error.

## 5.3 Component Names

A component name can either be the simple, "relative" name of a component or, within the command-line interface, a full component name, analogous to a relative filename or fully qualified, in other words, absolute filename.

The syntax for component names consists of a root component, specified by a colon (`:`), followed by the name of the child component of the root component, followed by a colon separator, followed by a child of the previous component, and so on. So an example component name is:

```
:src:bin:cat
```

**Note:** When specifying an open bracket on the command line, remember to preface it with a backslash character (`\`), so that the shell doesn't misinterpret it.

The user can specify a revision name in square brackets appended to a component name. For example, to acquire the revision named `rev1` of the component named `:src:bin:cat`, the user specifies the following component to the `acquire` command:

```
:src:bin:cat[rev1]
```

When the user appends a revision name to a component name, the user refers to a "consistent" instance of the revision, in other words, that named revision of the component.

A component name is a string of alphanumeric characters and colon characters with a maximum length of 128 characters, including any revision name appended to the component name.

**Note:** When acquiring the latest consistent revision of a component using the empty square bracket syntax, the user need not know the name of the revision to acquire it.

To obtain the latest “consistent” instance, in other words the latest revision, of a component, the user may specify the component name, appending empty square brackets after it. For example, to obtain the latest revision, say `rev5`, of the component named `:src:bin:cat`, the user could specify:

```
:src:bin:cat[]
```

When the user specifies the abbreviation colon (:) as a component argument to certain commands, the command operates on all of the components from the specified environment.

## 5.4 Target Names

When specifying a target object to a command, the user must provide a target name, followed by a percent sign (%), followed by the fully qualified, in other words, absolute pathname for the Makefile associated with the target.

A complete target name is a string of alphanumeric characters and a percent sign with a maximum length of 1024 characters.

## 5.5 Filenames

When specifying a filename to either the Browser or the command-line interface, in most cases the filename must be the fully qualified, in other words absolute pathname for the file.

A simple file name is a string of alphanumeric characters with a maximum length of 255 characters. An absolute pathname for a file is a series of simple file names, separated by slash characters (/), with a maximum length of 1024 characters.

## 5.6 Link Database Names

When specifying a link database object to a command, the user must provide the fully qualified name, in other words, the absolute pathname of the link database.

A simple link database name is a string of alphanumeric characters with a maximum length of 255 characters. An absolute pathname for a link database is a series of simple file names, separated by slash characters (/), with a maximum length of 512 characters.





# A

## Glossary

|               |     |
|---------------|-----|
| Glossary..... | 259 |
|---------------|-----|







# A

## Glossary

This glossary defines some of the terms included in this manual. For an alphabetized summary of the NSE commands, see Section 4.3, *Command Summaries*.

### acquire

The first step of the acquire/reconcile/resolve cycle, in which a user obtains access to a component hierarchy, which may include targets, files, and other NSE objects, within that user's environment. *See also* **acquire/reconcile/resolve cycle and environment**.

### acquire/reconcile/resolve cycle

The development cycle in which a developer accesses development objects, such as source files and associated libraries, within an isolated development environment (child), makes changes, tests, and merges the changes into the shared integration environment (parent). Sometimes developers iterate recursively through the cycle to resolve all the merge conflicts that occur. An analogous cycle takes place when users merge child integration environments into their release or integration environment parents. *See also* **acquire, development environment, integration environment, isolate, reconcile, and resolve**.

### activate

To ready an environment for use by accessing the virtual file system associated with that environment.

### administration

Maintenance of various aspects of the NSE, such as overall organization of the environment hierarchy and location of environments on machines. *See also* **bootstrap and installation**.

### application class

A set of applications that can display the same type of NSE application-specific key is an application class, for example `textedit`. Application classes may contain one or more applications. Applications may belong to one or more classes. *See*



|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                 | <i>also application registration, application-specific key, link, link database, link label, and link service.</i>                                                                                                                                                                                                                                                                                                             |
| <b>application registration</b> | The procedure by which an application indicates to the link service that it is available to display links of a specified application class. <i>See also application class, application-specific key, link, link database, link label, and link service.</i>                                                                                                                                                                    |
| <b>application-specific key</b> | The part of an NSE link that an application stores as a “handle” to the contents of that link. The application-specific key contains enough information for an application to display the object associated with that link. Each application-specific key is a variable-length, null-terminated, ASCII string. <i>See also application class, application registration, link, link database, link label, and link service.</i> |
| <b>architecture</b>             | A kind of machine hardware, especially the kind of central-processing unit, and system software associated directly with that kind of hardware, for example, Sun-3 or Sun-4. <i>See also variant.</i>                                                                                                                                                                                                                          |
| <b>ASK</b>                      | <i>See application-specific key.</i>                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>binary file</b>              | A file that includes non-printable ASCII characters, including all a.out files, in other words, most executable and object files. <i>See also object file.</i>                                                                                                                                                                                                                                                                 |
| <b>bootstrap</b>                | The process of putting a UNIX file and directory hierarchy under the control of the NSE. <i>See also administration and installation.</i>                                                                                                                                                                                                                                                                                      |
| <b>browse</b>                   | To look, or traverse a path through a hierarchy of environments and associated components, targets, files, and other NSE objects. <i>See also query.</i>                                                                                                                                                                                                                                                                       |
| <b>build</b>                    | To construct a version of a development object suitable for its intended use, such as constructing an executable file from source code. <i>See also compile.</i>                                                                                                                                                                                                                                                               |
| <b>child environment</b>        | An environment located just “below,” in other words created from a parent environment within an environment hierarchy. <i>See also environment and parent environment.</i>                                                                                                                                                                                                                                                     |
| <b>class</b>                    | <i>See application class.</i>                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>command-line interface</b>   | NSE commands accessible by typing to a shell prompt, or otherwise accessible through a UNIX shell, in contrast to operations                                                                                                                                                                                                                                                                                                   |

- accessible through a window-based tool designed especially for the NSE. *See also* **browse**.
- compile** To build an executable version of a program, or a program development module. Can involve use of a makefile. *See also* **build**.
- component** A component is a group of related development objects that constitute a functional unit. A typical low-level component consists of a set of objects: an executable program plus the source, include, object, and library files necessary to compile and link the executable program. A component can also include the program's design documentation, test data, and a test driver. Thus, a component gathers into a single unit everything needed to modify a functional piece of the larger system. Higher-level components group lower-level components together; usually, they correspond to subsystems or large programs.
- constrained file** A file that the NSE doesn't permit users within a given environment to alter because that file is built from files or objects that are not available within that environment.
- derived file** Any file that exists because developers build it from other files or NSE objects, such as .o object files (C programming language), executables, and libraries. *See also* **object file**.
- development environment** An environment without children in which developers make changes, build, and test development objects. Developers integrate changes made in development environments into the parent integration environment. *See also* **environment**, **integration environment**, and **isolation**.
- development object** *See* **object**.
- environment** An individual workspace for NSE developers that provides isolation and stability along with consistent naming of objects. Environments consist of a hierarchy of development objects and associated tools. Because environments provide an isolated workspace for developers, the developers can make changes to development objects without influencing each other's work. *See also* **development environment** and **integration environment**.
- executable set** A portion of a virtual file system hierarchy associated with one or more variants that acts as a stable set of executables necessarily visible within an environment hierarchy so as to derive objects. Revisions often include an executable set of the tools necessary to



|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                | recreate the executables stored in that revision. <i>See also</i> <b>executable file</b> <i>and</i> <b>variant</b> .                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>file</b>                    | <i>See</i> <b>constrained file</b> <i>or</i> <b>derived file</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>installation</b>            | The process of getting the NSE, or NSE extensions, off a tape and onto machines prior to bootstrapping and use of the NSE. <i>See also</i> <b>administration</b> <i>and</i> <b>installation</b> .                                                                                                                                                                                                                                                                                                                                          |
| <b>integration environment</b> | A mid-level environment, usually a child environment of a release environment, into which developers merge the changes they have made in various child development environments. <i>See also</i> <b>development environment</b> <i>and</i> <b>environment</b> .                                                                                                                                                                                                                                                                            |
| <b>isolation</b>               | A state that makes it possible for developers to make changes to development objects without causing changes in, or being influenced by changes in other developers' environments. <i>See also</i> <b>development environment</b> , <b>environment</b> , <i>and</i> <b>object</b> .                                                                                                                                                                                                                                                        |
| <b>leaf environment</b>        | <i>See</i> <b>development environment</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>link</b>                    | A connection between two objects made so that an application, or two applications can display each of the objects. For each object, a link contains an application-specific key, an application class, and a link label. <i>See also</i> <b>application class</b> , <b>application registration</b> , <b>application-specific key</b> , <b>link database</b> , <b>link label</b> , <i>and</i> <b>link service</b> . <i>Not related to search link</i> .                                                                                    |
| <b>link database</b>           | A database, or easy-access spot that stores links. <i>See also</i> <b>application class</b> , <b>application registration</b> , <b>application-specific key</b> , <b>link</b> , <b>link label</b> , <i>and</i> <b>link service</b> . <i>Not related to search link</i> .                                                                                                                                                                                                                                                                   |
| <b>link label</b>              | A null-terminated, human-readable, ASCII string, associated with a link, that contains identifying information a user can view for that link. <i>See also</i> <b>application class</b> , <b>application registration</b> , <b>application-specific key</b> , <b>link</b> , <b>link database</b> , <i>and</i> <b>link service</b> . <i>Not related to search link</i> .                                                                                                                                                                     |
| <b>link service</b>            | A mechanism that permits users or tools to connect one NSE object with another NSE object using a link between these objects within one, or between two applications. For example, a user can link a box representing a procedure on a design diagram with the code that implements that procedure. Similarly, a user can link the documentation for a procedure with the corresponding code. The NSE already provides user programs and a programmatic interface to access and traverse links. <i>See also</i> <b>application class</b> , |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <b>application registration, application-specific key, link, link database, and link label.</b> <i>Not related to search link.</i>                                                                                                                                                                                                                                                                                                                                                |
| <b>locking</b>               | A mechanism for preventing merges of NSE development objects, as in the locking of a source code file by one programmer to prevent other programmers from making changes to it. For NSE 1.0, the only implemented portion of locking is a mechanism ( <code>envlock</code> ) for preventing reconciliation from a child environment to its parent so that release engineers can isolate an environment for release distribution. <i>See also constrained file.</i>                |
| <b>merge</b>                 | <i>See reconcile.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>notification facility</b> | An NSE mechanism that permits developers to schedule tasks to occur as development objects change, more specifically when reconciling or preserving a component. Example notification tasks include sending an electronic mail message or performing a system build. Often used to let users or tools know when a developer finishes resolution of a particular development object into the shared development environment. <i>Not related to notifier.</i>                       |
| <b>object</b>                | The NSE delivers a set of development objects, including files, Makefile targets, and components. Users may construct a hierarchy of development objects within an environment by using components. Software requirements, design documents and drawings, test data and drivers, schedules, budgets, and staffing plans are available by extending the list of development object types. <i>See also component.</i>                                                               |
| <b>object file</b>           | A type of derived file that program compilers produce as an input to loaders. <i>See also derived file.</i>                                                                                                                                                                                                                                                                                                                                                                       |
| <b>parent environment</b>    | The environment "on top of" another environment, or from which a user created another environment within an environment hierarchy. <i>See also child environment and environment.</i>                                                                                                                                                                                                                                                                                             |
| <b>query</b>                 | A request to browse the contents of an environment, a component, a target, a file, or another object within the NSE. <i>See also browse.</i>                                                                                                                                                                                                                                                                                                                                      |
| <b>reconcile</b>             | A step in the acquire/reconcile/resolve cycle that delineates the differences between the hierarchy of objects in a child environment and its parent environment. These differences usually are a result of changes made by developers to objects in the child environment. The purpose of reconciling is to permit merging of these changes into the parent environment. If no one has changed the contents of the parent environment since a user acquired a development object |



hierarchy from it, that user can automatically merge the altered contents of the child environment into the parent environment. *See also acquire/reconcile/resolve cycle and environment.*

**registration**

*See application registration.*

**resolve**

A step in the acquire/reconcile/resolve cycle during which the user evaluates the difference between object hierarchies in a parent and child environment to decide what changes to incorporate into the new revision of the parent environment. *See also acquire/reconcile/resolve cycle and environment.*

**revision**

A snapshot of the current state of an environment and its contents made so that users can acquire components from that snapshot of the environment at a later time if necessary for bug fixes or support consultations. A revision can include a copy of the tools used to build its object files, ensuring that the user can reliably recreate object files despite the installation of new tools. *See also working context.*

**subcomponent**

A child component, or component “below” or “inside” another component within a component hierarchy. *See also component.*

**system administration**

*See administration.*

**variant**

One of the problems with developing software in a heterogeneous network is supporting multiple machine architectures from a single set of development objects. NSE variants allow users to compile object files for multiple architectures automatically from a single set of source files. Variants ensure that an object file destined to run on a particular machine always builds with the correct compiler, object files, and libraries. *See also architecture.*

**VCS**

*See version control system.*

**version control system**

A mechanism for recording the history of changes to all objects under its control, permitting reconstruction of any previous version of an object at any time. As an example, when a user integrates a package for drawing data flow diagrams with the NSE, the NSE maintains versions of these drawings. To minimize consumption of disk space, the NSE Version Control System whenever possible stores only the differences in successive versions. *See also version.*

**working context**

The set of objects currently accessible to a user within an environment. *See also revision.*

---

# Index

## A

- acquire 164, 261
- acquire-from 222
- Acquire... 50
- acquire/reconcile/resolve cycle 261
- action 222
- activate 168, 261
- activated environment
  - command prompt 151
- Add Member... 53
- administration 261
- administrator xxi
- All Components 55
- ancestor
  - common 132
- append 170
- Append... 56
- application class 261
- application registration 262
- application-specific key 262
- architecture 262
- ASK 262
- audience xix
- automerging 132

## B

- binary file 262
- bootstrap 173, 262
- browse 262
- Browser 1
  - Acquire... operation 50
  - Add Member... operation 53
  - All Components operation 55
  - Append... operation 56
  - Checkin... operation 58

- Checkout operation 60
- Child Envs operation 62
- Clean Up operation 63
- comments 45
- component menu 21
- component symbol 9
- Contents operation 64
- Dbxtool operation 65
- Delete=> operation 67
- Descriptors operation 68
- Diffs=> operation 69
- Dismiss operation 71
- display menu 36
- display subwindow 5
- Emacs operation 73
- environment menu 18
- environment symbol 8
- file menu 25
- file symbol 11
- frame menu 40
- Get Version... operation 75
- History operation 76
- Kill operation 78
- Latest operation 79
- legend menu 35
- legend subwindow 4
- link database menu 28
- link database symbol 12
- List History operation 89
- List My Envs operation 90
- List Other's Envs... operation 91
- List Public Envs operation 92
- List Source Files operation 93
- List=> operation 80
- Lock operation 94
- log item menu 33
- log menu 38



- log subwindow 6
- make target symbol 10
- Make=> operations 95
- menu items 49
- notification menu 32
- Notifications operation 97
- Other Target... 98
- Parent Envs operation 99
- Preserve... operation 100
- Previous Queries operations 101
- queries 16
- Reconcile=> operation 102
- Recover=> operation 103
- Register Notification... operation 104
- Remove operation 106
- Rename... operations 107
- Resolve=> operation 111
- Resync=> operation 112
- Root Component operation 113
- selection menus 17
- Shelltool operation 114
- Show Output operation 115
- start up 2
- subwindow, display 5
- subwindow, legend 4
- subwindow, log 6
- subwindows 4
- symbols 7
- target menu 23
- target symbol 10
- Textedit operation 116
- Two Versions.... 118
- Uncheckout operation 119
- Unlock operation 120
- Unregister operation 121
- using text editor with 126
- variant menu 30
- variant symbol 13
- Variants operation 122
- Version... operation 123
- Vi operation 124
- window header 2
- window menus 35
- You Specify... operation 125
- Browser menus 15
- Browser operations 49
- build 262
- button
  - Cancel 43
  - Execute 43

## C

- Cancel 143
- Cancel graphical button 43
- Checkin... 58
- Checkout 60
- child environment 262
- Child Envs 62
- class 262
- Clean Up 63
- command prompt
  - activated environments 151
- command summaries 163
- command-line interface 152, 262
- comments 45
- common ancestor 132
- compile 263
- component 263
- component menu 21
- component names 256
- component symbol 9
- computer-assisted software engineering xxi
- Confirm All 143
- confirmation windows 46
- constrained file 263
- control point 199
- Contents 64
- Create 143
- create\_example 153
- customer support xxi

## D

- Database 143
- database
  - link 140
- dbresolve 135, 175
- Dbxtool 65
- Delete 143
- Delete => 67
- derived file 263
- describe\_comps 179
- Descriptors 68
- development
  - other xxi
  - software xix
- development environment 263
- development object 263
- dialog boxes 42
- Diffs => 69
- Dismiss 71

Display 143  
 display menu 36  
 display subwindow 5  
 documentation set xxii  
 documentation writing xx

## E

Emacs 73  
 End Link 143  
 entity 222  
 environment 263  
     activated command prompt 151  
 environment menu 18  
 environment names 255  
 environment symbol 8  
 envlock 222  
 envunlock 222  
 executable set 263  
 Execute graphical button 43

## F

file 264  
 file menu 25  
 file symbol 11  
 filenames 257  
 fileresolve 133, 181  
 frame menu 40

## G

Get Version... 75  
 Getting Started with UNIX  
     Beginner's Guide xix  
 Glossary 261  
 glyphs 132, 175

## H

History 76  
 history windows 47

## I

installation 264  
 integration environment 264  
 isolation 264

## K

Kill 78

## L

Latest 69, 79  
 leaf environment 264  
 legend menu 35  
 legend subwindow 4  
 link 139, 264  
 link database 264  
     names 257  
 link database menu 28  
 link database symbol 12  
 link databases 140  
 link label 264  
 Link Service 139  
 link service 264  
 Link Service  
     Cancel operation 143  
     Confirm All operation 143  
     Create operation 143  
     Database operation 143  
     Delete operation 143  
     Display operation 143  
     End Link operation 143  
     environment 140  
     operations 143  
     Set operation 143  
     Start Link operation 143  
     starting up a 141  
     user interface 141  
     window header 141  
 linksvc 141, 186  
 List => 80  
 List History 89  
 List My Envs 90  
 List Other's Envs... 91  
 List Public Envs 92  
 List Source Files 93  
 Lock 94  
 locking 265  
 log item menu 33  
 log menu 38  
 log subwindow 6  
 lvcs 187

## M

Make => 95



- make target symbol 10
- manager xix
- menu
  - component 21
  - display 36
  - environment 18
  - file 25
  - frame 40
  - legend 35
  - link database 28
  - log 38
  - log item 33
  - notification 32
  - selection 17
  - target 23
  - variant 30
  - window 35
- menu items
  - Browser 49
- menus
  - Browser 15
- merge 265
- merged 189
- merging with windows 132

## N

- naming specifications 255
- Network Software Environment
  - Installation and Administration xix, xxi, xxii
  - Reference Manual xxii
- notification
  - action 222
  - entity 222
  - request 222
- notification facility 222, 265
- notification menu 32
- Notifications 97
- nsebrowse 193
- nsecomp 195
- nsecontrolpt 199
- nseenv 201
- nseexecset 213
- nseivp 220
- nsemark 221
- nsenotify 222
- nseymlink 226
- nsevar 229

## O

- object 265
- object file 265
- operations
  - Browser 49
- options specification window 44
- Other Target... 98
- output windows 48

## P

- parallel\_update 157
- parameters specification window 44
- parent environment 265
- Parent Envs 99
- popup windows 42
- preserve 233
- Preserve... 100
- Previous Queries => 101
- programmer xx
- project leader xx

## Q

- queries
  - Browser 16
- query 265

## R

- reconcile 236, 265
- Reconcile => 102
- reconcile-to 222
- Recover => 103
- Register Notification... 104
- registration 266
- release management xx
- Remove 106
- Rename... 107
- request 222
- resolution of differences 175
- resolve 131, 241, 266
  - sample 133
- Resolve => 111
- resolving objects 131
- resync 243
- Resync => 112
- resync-from 222
- revision 266
- Root Component 113

## S

- selection menus 17
- Set 143
- Shelltool 114
- Show Output 115
- software development xix
- Start Link 143
- starting up a Link Service 141
- starting up the Browser 2
- subcomponent 266
- subwindow
  - display 5
  - legend 4
  - log 6
- subwindows
  - Browser 4
- symbol
  - component 9
  - environment 8
  - file 11
  - link database 12
  - make target 10
  - target 10
  - variant 13
- symbols
  - Browser 7
- system administration xxi, 266

## T

- target menu 23
- target names 257
- target symbol 10
- text entry box 42
- Textedit 116
- Two 69
- Two Versions... 69, 118

## U

- Uncheckout 119
- Unlock 120
- Unregister 121
- using text editor with Browser 126

## V

- variant 266
- variant menu 30
- variant names 256

- variant symbol 13
- Variants 122
- VCS 266
- vcs 246
- version control system 266
- Version... 69, 123
- Vi 124

## W

- window
  - confirmation 46
  - history 47
  - options specification 44
  - output 48
  - parameters specification 44
  - popup 42
- window header
  - Browser 2
  - Link Service 141
- window menus 35
- windows
  - merging with 132
- Windows and Window-Based Tools
  - Beginner's Guide 45, 47, 48
- working context 266
- writer xx

## Y

- You Specify... 125





---

## Revision History

| Version | Date           | Comments                  |
|---------|----------------|---------------------------|
| A       | 11 March 1988  | First release of NSE 1.0. |
| A       | 7 October 1988 | First release of NSE 1.1. |









**Corporate Headquarters**

Sun Microsystems, Inc.  
2550 Garcia Avenue  
Mountain View, CA 94043  
415 960-1300  
TLX 287815

**For U.S. Sales Office**

locations, call:

800 821-4643

In CA: 800 821-4642

**European Headquarters**

Sun Microsystems Europe, Inc.  
Sun House  
31-41 Pembroke Broadway  
Camberley  
Surrey GU153XD  
England  
0276 62111  
TLX 859017

**Australia:** 61-2-436-4699

**Canada:** 416 477-6745

**France:** (1) 46 30 23 24

**Germany:** (089) 95094-0

**Japan:** (03) 221-7021

**The Netherlands:** 02155 24888

**UK:** 0276 691297

**Europe, Middle East, and Africa,**

**call European Headquarters:**

0276 62111

**Elsewhere in the world,**

**call Corporate Headquarters:**

415 960-1300

Intercontinental Sales

